

OBJECT ORIENTED TURING

REFERENCE MANUAL

**Seventh Edition
Version 1.0**

**R. C. Holt
with
Tom West**

**Holt Software Associates Inc.
203 College Street, Suite 305
Toronto, Ontario M5T 1P9**

www.holtsoft.com

(416) 978-8363

ISBN 0-921598-35-1

All right reserved. No part of this book may be reproduced in any way, or by any means, without permission of the author.

© 1999 by the authors

ISBN 0-921598-35-1

Printed in Canada

For more information about Turing such as textbooks, resources and additional materials, check Holt Software's web site located at:

<http://www.holtsoft.com/turing>

Table of Contents

CHAPTER ONE : INTRODUCTION	1
Language Evolution	1
Other Material about Turing	1
Two Main Implementations: Turing and OOT	2
Using Turing and OOT for Teaching	2
Design of the Language	3
Dirt and Danger	3
Implementing Turing and OOT	4
Basic Concepts in Turing	4
Compile-Time Expressions	6
Documentation of Language Features	6
OOT Predefined Units	7
Changes to OOT	8
CHAPTER TWO : OOT FOR WINDOWS	9
Installing OOT for Windows	9
Minimum System Requirements	9
Installing Under Windows 3.1	9
De-installing Under Windows 3.1	12
Enabling Virtual Memory Under Windows 3.1	12
Changing Monitor Resolution Under Win 3.1	12
Installing Under Windows 95	14
MS Windows Drivers for WinOOT	15
Using OOT for Windows	16
Starting WinOOT	16
WinOOT Environment	17
OOT Windows - Control Panel	18
OOT Windows - Directory Viewer	20
OOT Windows - Editor Window	21
OOT Windows - Error Viewer	22
OOT Windows - Run Window	23
OOT Windows - Debugger Window	24
Editing Text - Keyboard Editing Commands	26
Editing Text - Selection, Cutting and Pasting	27
Editing Text - Searching and Replacing Text	27
Running Programs - Input and Output Redirection	29
Running Programs - Programs with Multiple Parts	31
Menu Commands - Control Panel	32
Menu Commands - Directory Viewer	34
Menu Commands - Editor Window	36
Debugging - Stepping and Tracing	40
Debugging - Showing Variables	42
Tutorial	45

CHAPTER THREE : OOT FOR MACINTOSH	49
Installing OOT for Macintosh	49
Minimum System Requirements	49
Installing Under Macintosh	49
De-installing OOT for Macintosh	52
Using OOT for Macintosh	53
Starting MacOOT	53
MacOOT Environment	53
MacOOT Windows - Editor Window	55
MacOOT Windows - Run Window	56
MacOOT Windows - Debugger Window	57
MacOOT Windows - Debugging Controller	59
MacOOT Windows - Error Viewer	60
Editing Text - Keyboard Editing Commands	61
Editing Text - Selection, Cutting and Pasting	62
Editing Text - Searching and Replacing Text	62
Running Programs -	64
Input and Output Redirection	64
Running Programs -	66
Programs with Multiple Parts	66
Menu Commands	67
Debugging - Stepping and Tracing	72
Showing Variables	74
Tutorial	77
CHAPTER FOUR : OOT FOR DOS	81
Installing OOT for DOS	81
Minimum System Requirements	81
Installing Under DOS	81
De-installing Under DOS	85
Using OOT for DOS	86
Starting DOS OOT	86
Selecting a Menu Item in DOS OOT	87
Loading a Program in DOS OOT	88
Editing a Program in DOS OOT	90
Saving a Program in DOS OOT	92
Running a Program in DOS OOT	93
To Exit DOS OOT	93
OOT for DOS Editor Reference	94
The Menu Commands:	94
The Window Commands	99
Selecting Text	99
Mouse Movement	100
Dialog Boxes	101
The Pick File	114
Start Up Options	115
Compilation to Executable File	119
Input and Output Redirection	119

Summary	120
CHAPTER FIVE : THE GUI MODULE	123
Introduction	123
Terms	125
Example Programs	125
General Principles of the GUI Library	125
Active and Passive Widgets	127
Keyboard Shortcuts	130
Background Color	130
Widget Sizes	131
Positioning Text Labels (Aligning Labels with Widgets)	132
Canvases	133
Multiple Windows	135
The GUI Library Internals	136
GUI Module Routines Summary	138
Widgets - Common Routines	141
Widgets - Buttons	142
Widgets - Check Boxes	143
Widgets - Radio Buttons	145
Widgets - Picture Buttons	146
Widgets - Picture Radio Buttons	147
Widgets - Sliders	148
Widgets - Scroll Bars	150
Widgets - Canvases	152
Widgets - Text Fields	154
Widgets - Text Boxes	155
Widgets - Lines	156
Widgets - Frames	157
Widgets - Labelled Frames	158
Widgets - Labels	159
Widgets - Pictures	160
Widgets - Menus	161
Widgets - General Routines	162
CHAPTER SIX : IMPORTING GRAPHICS INTO TURING	165
Introduction	165
Importing BMP Images in WinOOT	165
Pic.ScreenLoad	166
Pic.ScreenSave	167
Importing PICT Images in MacOOT	167
Pic.ScreenLoad	167
Pic.ScreenSave	168
Importing PCX Images in DOS OOT	168
PCX2TM2	168
Pic.ScreenLoad	170
Pic.ScreenSave	171

CHAPTER SEVEN : LANGUAGE FEATURES

173

abs	absolute value function	159
addr	address of a variable	160
addressint	type	160
all	all members of a set	161
and	operator	162
anyclass	the ancestor of all classes	163
arctan	arctangent function (radians)	164
arctand	arctangent function (degrees)	164
array	type	165
assert	statement	168
assignability	of expression to variable	169
assignment	statement	170
begin	statement	171
bind	declaration	172
bits	extraction	172
body	declaration	174
boolean	true-false type	175
break	debugger pause statement	176
buttonmoved	has a mouse event occurred	178
buttonwait	get a mouse event procedure	180
case	selection statement	181
catenation (+)	joining together strings	182
ceil	real-to-integer function	183
char	type	184
char(n)	type	185
cheat	type cheating	187
checked	compiler directive	189
chr	integer-to-character function	189
class	declaration	190
clock	millisecs used procedure	195
close	file statement	195
cls	clear screen graphics procedure	196
collection	declaration	197
color	text color graphics procedure	198
colour	text colour graphics procedure	198
colorback	background color procedure	199
colourback	background colour procedure	199
comment	remark statement	200
comparisonOperator		200
Concurrency		201
Concurrency.empty		202
Concurrency.getpriority		202
Concurrency.setpriority		203

Concurrency.simutime	203
condition declaration	204
Config	206
Config.Display	206
Config.Lang	207
Config.Machine	208
const constant declaration	209
constantReference use of a constant	210
cos cosine function (radians)	210
cosd cosine function (degrees)	211
date procedure	211
declaration create a variable	212
deferred subprogram declaration	213
delay procedure	214
Dir	215
Dir.Change	216
Dir.Close	216
Dir.Create	217
Dir.Current	218
Dir.Delete	218
Dir.Get	219
Dir.GetLong	220
Dir.Open	222
div integer truncating division operator	223
Draw	223
Draw.Arc	224
Draw.Box	225
Draw.Cls	226
Draw.Dot	226
Draw.Fill	227
Draw.FillArc	228
Draw.FillBox	229
Draw.FillMapleLeaf	230
Draw.FillOval	231
Draw.FillPolygon	232
Draw.FillStar	233
Draw.Line	234
Draw.MapleLeaf	234
Draw.Oval	235
Draw.Polygon	236
Draw.Star	237
Draw.Text	238
empty condition function	240
enum enumerated type	240

enumeratedValue	enumerated value	241
eof	end-of-file function	242
equivalence	of types	243
erealstr	real-to-string function	244
Error		245
Error.Last		246
Error.LastMsg		246
Error.LastStr		247
Error.Msg		248
Error.Str		249
Error.Trip		249
ErrorNum		250
Exceptions		250
exit	statement	250
exp	exponentiation function	251
explicitCharConstant	character literal	252
explicitConstant	literal	252
explicitIntegerConstant	integer literal	253
explicitRealConstant	real literal	254
explicitStringConstant	string literal	255
explicitTrueFalseConstant	boolean literal	256
expn	expression	257
export	list	258
external	declaration	260
false	boolean value (not true)	261
fetcharg	fetch argument function	261
File		262
File.Copy		263
File.Delete		264
File.DiskFree		264
File.Exists		265
File.Rename		266
File.Status		267
flexible	array initialization	268
Font		270
Font.Draw		271
Font.Free		272
Font.GetName		273
Font.GetSize		273
Font.GetStyle		274
Font.Name		275
Font.New		275
Font.Sizes		278
Font.StartName		279

Font.StartSize	279
Font.Width	280
for statement	281
fork statement	283
forward subprogram declaration	284
frealstr real-to-string function	285
free statement	286
function declaration	287
functionCall	288
get file statement	289
getch get character procedure	293
getchar get character function	294
getenv get environment function	295
getpid get process id function	295
getpriority function	296
GUI	296
GUI.AddLine	296
hasch has character function	393
id (identifier) name of an item in a program	394
if statement	394
#if used for conditional compilation	396
implement clause	397
implement by clause	399
import list	399
in member of a set	401
include source files	402
index find pattern in string function	403
indexType	404
indirection operator (@)	404
infix operator	405
inherit inheritance clause	407
init array initialization	409
Input	410
Input.getch	411
Input.getchar	412
Input.hasch	413
Input.Pause	413
int integer type	414
intn n-byte integer type	415
intreal integer-to-real function	416
intstr integer-to-string function	416
invariant assertion	417
Joystick	418
Joystick.GetInfo	418

Limits	420
ln natural logarithm function	421
locate procedure	421
locatexy graphics procedure	422
loop statement	423
lower bound	424
Math	424
max maximum function	425
maxcol maximum column function	426
maxcolor graphics function	426
maxint maximum integer function	427
maxnat maximum natural number function	427
maxrow maximum row function	428
maxx graphics function	428
maxy graphics function	429
min minimum function	430
minint minimum integer function	430
minnat minimum natural number function	431
mod modulo operator	432
module declaration	433
monitor declaration	436
Mouse	439
Mouse.ButtonChoose	439
Mouse.ButtonMoved	440
Mouse.ButtonWait	442
Mouse.Hide	444
Mouse.Show	445
Mouse.Where	446
Music	447
Music.Play	447
Music.PlayFile	448
Music.Sound	449
Music.SoundOff	450
named type	451
nargs number of arguments	451
nat natural number type	452
natn n-byte natural number type	453
natreal natural number to real function	454
natstr natural-number-to-string function	454
Net	455
All subprograms in the Net unit are exported qualified (and thus must be prefaced with " Net. ").	455
nil pointer to a collection	468

not true/false (boolean) operator	469
objectclass of a pointer	469
opaque type	470
open file statement	470
or operator	473
ord character-to-integer function	474
palette graphics procedure	474
parallelget parallel port function	475
parallelput parallel port procedure	476
paramDeclaration parameter declaration	477
pause statement	479
PC	480
PC.InPortWord	481
PC.Interrupt	482
PC.InterruptSegs	483
PC.OutPort	484
PC.OutPortWord	485
PC.ParallelGet	486
PC.ParallelPut	487
PC.SerialGet	488
PC.SerialHasch	489
PC.SerialPut	490
pervasive declaration modifier	491
Pic	492
Pic.FileNew	493
Pic.Free	495
Pic.New	495
Pic.Save	496
Pic.ScreenLoad	498
Pic.ScreenSave	499
play procedure	500
playdone function	501
pointer type	501
post assertion	504
pre assertion	504
precedence of operators	505
pred predecessor function	507
prefix operator	507
procedure declaration	508
procedureCall statement	510
process declaration	511
program a (main) program	512
put statement	513
quit fail statement	515

Rand	516
rand random real number procedure	517
Rand.Int	518
Rand.Reset	520
Rand.Seed	521
Rand.Set	521
randomize procedure	522
read file statement	523
.c.read file statement	523
real the real number type	524
real n n-byte real number type	525
realstr real-to-string function	526
record type	527
register use machine register	528
rem remainder operator	528
repeat make copies of string function	529
result statement	529
return statement	530
RGB	531
RGB.AddColor	531
RGB.GetColor	532
RGB.maxcolor	533
RGB.SetColor	534
scalar type	535
seek (file) statement	536
self pointer to current object	537
separator between tokens in a program	537
serialget serial port function	538
serialput serial port procedure	539
set type	540
setConstructor	541
setpriority procedure	541
setscreen graphics procedure	542
shl shift left operator	545
shr shift right operator	545
sign function	546
signal wake up a process statement	546
simutime simulated time function	547
sin sine function (radians)	548
sind sine function (degrees)	548
sizeof size of a type	549
sizepic graphics function	549
skip used in get statement	550
skip used in put statement	551

sound statement	551
Sprite	552
All subprograms in the Sprite unit are exported qualified (and thus must be prefaced with " Sprite. ").	552
standardType	560
statement	560
statementsAndDeclarations	562
Str	562
Stream	563
Stream.Flush	564
Stream.FlushAll	565
string comparison	565
string type	566
strint string-to-integer function	567
strintok string-to-integer function	568
strnat string to natural number function	568
strnatok string to natural number function	569
strreal string-to-real function	569
strrealok string-to-real function	570
subprogramHeader	570
subprogramType	571
subrangeType	572
substring of another string	573
succ successor function	574
Sys	575
Sys.FetchArg	576
Sys.GetEnv	577
Sys.GetPid	577
Sys.Nargs	578
sysclock millisecs used procedure	579
system statement	579
tag statement	581
takepic graphics procedure	582
tell file statement	583
Text	584
Text.Cls	585
Text.Color	585
Text.Colour	585
Text.ColorBack	586
Text.ColourBack	586
Text.Locate	587
Text.LocateXY	588
Text.maxcol	589
Text.maxrow	589

Text.WhatChar	590
Text.WhatCol	591
Text.WhatColor	591
Text.WhatColour	591
Text.WhatColorBack	592
Text.WhatColourBack	592
Text.WhatRow	593
Time.DateSec	595
Time.Delay	596
Time.Elapsed	596
Time.ElapsedCPU	597
Time.PartsSec	598
Time.Sec	598
Time.SecDate	599
Time.SecParts	600
time time of day as a string procedure	600
token in input	601
true boolean value (not false)	602
TypeConv	602
type declaration	603
typeSpec type specification	604
unchecked compiler directive	605
union type	606
unit file containing module, monitor, or class	607
unqualified export	609
upper bound	609
var declaration	610
variableReference use of a variable	611
View	612
View.ClipAdd	613
View.ClipOff	614
View.ClipSet	614
View.GetActivePage	615
View.GetVisiblePage	616
View.maxcolor	617
View.maxcolour	617
View.maxx	618
View.maxy	618
View.Set	619
View.SetActivePage	622
View.SetVisiblePage	623
View.WhatDotColor	624
Text.WhatDotColour	624
wallclock seconds since 1/1/1970 procedure	626

whatcol	cursor position function	626
whatcolor	text color graphics function	627
whatcolorback	color of background function	627
whatdotcolor	graphics function	628
whatpalette	graphics function	629
whatrow	cursor position function	629
whattextchar	graphics function	630
whattextcolor	graphics function	631
whattextcolorback	graphics function	631
Window		632
Window.Close		633
Window.GetActive		634
Window.GetPosition		634
Window.GetSelect		635
Window.Hide		636
Window.Open		636
Window.Select		637
Window.Set		638
Window.SetActive		639
Window.SetPosition		640
Window.Show		641
write	file statement	641
xor	exclusive "or" operator	642

APPENDICES

Appendix A	Predefined Subprograms and Keywords	693
Appendix B	OOT Predefined Subprograms	696
Appendix C	Turing Predefined Subprograms	750
Appendix D	Operators	775
Appendix E	File Statements	778
Appendix F	Control Constructs	779
Appendix G	IBM PC Keyboard Codes	780
Appendix H	Turing (IBM PC) Character Set	782

Acknowledgments

T.L. West has added extensive materials to this manual since the original edition. C. Stephenson has greatly contributed in her role as editor of the manual. Brenda Kosky created the cover design.

J.R.Cordy, the co-designer of the Turing Language, is acknowledged as the source of much of the material in this Manual. S.G. Perelgut contributed the Introductory section of this manual.

Technical Questions

Any technical problems or questions should be directed to our technical support line at Holt Software Associates. The number for this support line is (416) 978-8363 or 1-800-361-8324. Holt Software's web site, containing a wealth of Turing information including free downloadable resources, is located at:

<http://www.holtsoft.com/turing>

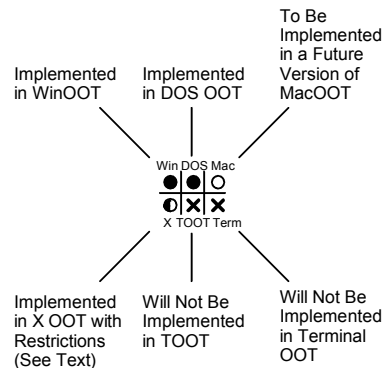
Legend

On the upper right of each predefined subprogram is a little symbol that describes whether the subprogram is available under a version of Object Oriented Turing. The versions of OOT referred to here are:

Win	Object Oriented Turing for Windows version 3.0
DOS	Object Oriented Turing for DOS version 2.5
Mac	Object Oriented Turing for Macintosh version 1.5
X	Object Oriented Turing for X Windows version
TOOT	Command line version Object Oriented Turing version
Term	This version of Object Oriented Turing does not exist at the time of this writing. It is included for completeness and refers to a version written for ASCII terminals under UNIX.

The symbols are as follows:

- This feature is currently implemented.
- This feature is not implemented at the time of writing. Check the release notes provided with your software to see if this feature has been implemented.
- ◐ This feature is implemented, but there are restrictions to its use. Consult the subprogram description for more details.
- ✕ This feature will not be implemented. This usually applies to predefined subprograms that don't make sense for a particular environment (such as the **Window** subprograms in a non-window environment such as DOS OOT, or a pixel graphics subprogram under a command line environment such as TOOT).



Chapter 1

Introduction

Object Oriented Turing is a high-level programming language that is easy to learn. This Manual has been written to provide easily accessible answers to technical questions about the language. It includes a list of the language features arranged in alphabetic order.

This Manual is directed to readers who have already been introduced to Turing or know a related language, such as Pascal, Basic or C and have some familiarity with operating systems such as UNIX.

While there are two languages currently available, Turing and Object Oriented Turing, this manual will, in most cases, abbreviate Object Oriented Turing to Turing. When referring to the original language Turing, it will use "Original Turing".

Language Evolution

Original Turing, as originally designed in 1982, is a Pascal-like language. It has been used extensively for teaching from grade 8 through graduate school. Original Turing as extended with graphics and binary file input/output is currently supported on IBM PC compatibles using MS-DOS and Apple Macintoshes. Object Oriented Turing (often abbreviated OOT or just Turing), was developed in the late 80's. It runs on IBM PC compatibles running Microsoft Windows or MS-DOS, Apple Macintoshes and UNIX/Linux under X Windows. Object Oriented Turing extends Original Turing with two sets of features. The first includes concurrency as well as systems programming features that allow the language to be used in place of C. The second is the object oriented features, which include classes, inheritance and polymorphism.

Other Material about Turing

Holt Software Associates Inc. also publishes J.N.P. Hume's *Turing Tutorial Guide* and *Problem Solving and Programming in Turing* which provide a beginner level introduction to the language using many examples. Advanced books on the language include *An Introduction to Computer Science Using the Turing Programming Language*, *Programming: Concepts and Paradigms*, and *Data Structures: An Object-Oriented Approach*. For more information on books published by Holt Software, check Holt Software's web site <http://www.holtsoft.com/turing>.

For the computer science expert, the book *The Turing Programming Language: Design and Definition* [Prentice-Hall publishers] provides detailed information about how and why Original Turing was designed. That book also contains both the *Turing Report*, which is the official defining document for the Original Turing language, and the formal (mathematical) definition of the language. The technical article *The Turing Programming Language*, appearing the Communications of the Association for Computing Machinery in December 1988, provides the best overview of Original Turing and its extension to Turing Plus.

Two Main Implementations: Turing and OOT

There are two main implementations of Turing, one called Original Turing that runs on IBM PC compatibles and Apple Macintoshes, and the other called Object Oriented Turing (OOT) that runs on UNIX under X Windows, and IBM PC compatibles running Microsoft Windows or DOS and the Apple Macintosh. Original Turing proper supports graphics features and binary input/output (read, write, etc.). OOT supports all the features of Original Turing as well as concurrency and systems programming features and object oriented features (classes, inheritance and polymorphism).

There are sets of features that are available in some versions of Original Turing but not in others, as annotated by x's in the following table. With the descriptions in this Manual, there are annotations such as [Pixel graphics only] that give the feature set to which the description belongs. This table lists an additional version: TOOT (Terminal OOT, a command line form of OOT available only in UNIX).

FEATURE SET

VERSION	Char Graphics	Pixel Graphics	Sound	UNIX	OOT
WinOOT		x	x		x
MacOOT		x	x		x
DOS OOT	x	x	x		x
Unix X OOT		x		x	x
TOOT				x	x

Using Turing and OOT for Teaching

Original Turing and OOT are ideal for teaching introductory Computer Science concepts and good programming habits. OOT is also ideal for teaching higher level concepts such as object oriented programming, programming in the large and concurrency. The concurrency and systems programming features of OOT are ideal for teaching systems programming (including operating systems and compilers). These features support the functionality and performance of C but eliminate the need for dealing with C's uncontrolled crashes.

Design of the Language

The Turing language has been designed to be a general purpose language, meaning that it is potentially the language of choice for a wide class of applications. Its convenience and expressive power make it particularly attractive for learning and teaching. Because of its clean syntax, Turing programs are relatively easy to read and write.

Turing helps programmers write more reliable programs by eliminating or constraining error-prone language features. For example, Turing eliminates the **goto** statement. It also provides many compile-time and run-time checks to catch bugs before they lead to disaster. These checks guarantee that a Turing program behaves according to specification. If it does not, a warning message is given.

Turing has been designed to eliminate the security problems of languages such as Pascal. For example, Turing's variant records (unions) have an explicit tag field that determines the active variant of the record. Run-time checks guarantee that the program can never access fields that have been assigned in a different variant. In principle, a Turing compiler prevents function *side effects* (changing values outside of the function) and *aliasing* (having more than one way to modify a given value). However, existing implementations do not enforce these restrictions.

Turing has modules, which serve as information hiding units. These modules are *objects* in the sense of "object-oriented programming". They allow the programmer to divide the program into units that have precisely controlled interfaces. OOT extends Turing's modules so they can have multiple instances. This extension, called classes, supports inheritance (called expansion) and polymorphism (overriding subprograms in expansions).

SmallTalk, one of the best known object oriented languages, considers that primitive items, such as integers, are "objects". This requires the beginner to unlearn classical mathematical concepts. OOT avoids this puritanical approach by considering that the integers and other well understood concepts are exactly those of mathematics. C++, another well known object oriented language, grafts classes into a structure that requires the beginning programmer to deal with source inclusion, conditional compilation, header files and a linkage editor. OOT avoids this complication by using classes and modules as the language's compilation units. The result, in OOT, is that programming in the large becomes object oriented and easy for the beginner (as well as the expert). OOT does not support automatic recovery of object space (garbage collection) or operator overloading.

Dirt and Danger

OOT contains a full set of systems programming features. This includes features that provide full access to the underlying run-time representation. While this access is needed (only occasionally!) by the systems programmer, it comes at a great cost in terms of portability to other machine platforms, reliability and productivity. As a result, the design of these features has isolated them so their use can be controlled.

Dirt. Language features that have implementation-dependent results are called *dirty*. For example, taking the address of a variable or inspecting the internal representation of a real number are both dirty actions. Features such as these are marked as **Dirty** in the

Manual. In the absence of dirty actions, a program can be expected to run with the same results on different platforms, for example on both IBM PC compatibles and Macintoshes.

Danger. Language features that allow changes to arbitrary machine memory, or that can cause an uncontrolled crash are called *dangerous*. For example, storing a value at a specified machine address or where a pointer points without checking the validity of the pointer are dangerous actions. Features such as these are marked as **Dangerous**. A programmer should resort to dangerous actions only when this is obviously necessary and the inherent loss in security can be justified.

In low level languages such as C, it is not possible to master even strings and arrays without dealing with the dangerous concepts of machine addresses. In C there is no syntactic differentiation of dirty or dangerous features. As a result, it is difficult to control program quality.

Implementing Turing and OOT

Original Turing was designed so it could be supported by either compilers or interpreters. At present (1999) the Original Turing system used on IBM PC compatibles and Apple Macintoshes is called an interpreter, although technically speaking it is a compiler that generates pseudo-code.

The OOT system is based on the Original Turing interpreter. Although a compiler is under consideration for OOT, at the present time, OOT is supported by a pseudo-code system. The OOT system makes extensive use of the GUI (graphical user interface) facilities the platforms it runs under to provide the user with an extremely rapid edit-and-test cycle, even when the program consists of a large number of classes and modules. In the spirit of object oriented programming, the OOT system supports browsing of reusable classes and rapid prototyping.

The Original Turing and OOT compilers and interpreters are all written in Turing Plus, a systems programming language developed from Turing.

Basic Concepts in Turing

Like most programming languages, Turing has variables whose values are changed by assignment statements. Example:

```
var i : int      % This declaration creates variable i
i := 10          % This assignment sets the value of i to 10
```

This example uses comments that begin with a percent sign (%) and which end with the end of the line. The names of items, such as *i*, are *identifiers*.

A *named constant* is a named item whose value cannot change. In the following, *c* and *x* are named constants:

```
const c := 25
const x := sin ( y ) ** 2
```

In this example, *c*'s value is known at compile-time, so it is a *compile-time* (or *manifest*) value. In the Pascal language, all named constants are compile-time values, but in Turing these values may be computed at run-time, as is the case for *x* in this example.

An *explicit constant* (or *literal constant*) is a constant that denotes its own value. For example, 27 is an explicit constant and so is "Hello".

In a Turing program, each named item such as a variable has a lifetime, which is called its *scope*. The lifetime of the item begins with its declaration and lasts to the end of the construct (such as a procedure or a loop) in which it is declared. More detail can be found under *declaration* in the main text. Turing's *scope rules* tell you where an item can be used. These rules are very similar to rules in Pascal.

Turing, like Pascal, has two kinds of subprograms, called *procedures* and *functions*. Procedures can be thought of as named sequences of statements, and functions can be thought of as operators that map values to values.

Turing allows you to declare an array whose size is not known until run time; these are called *dynamic arrays*. This is in contrast to languages such as Pascal in which the sizes of arrays must be known at compile time.

Turing can be thought of either as an *algorithmic language*, in which you write algorithms that will not necessarily be run on a computer, or as a *programming language* whose purpose is to be used on a computer. In the former case, the language is called *Ideal Turing* and is a mathematical notation, much as algebra and arithmetic are mathematical notations. In *Ideal Turing*, numbers have perfect accuracy (this is the nature of pure mathematics) and there is no limit on the size of a program or its data. By contrast, *Implemented Turing* is a language that you can use on actual computers.

Implemented Turing (which we usually call simply Turing) approximates Ideal Turing as closely as is practical. However, its integers have a fixed maximum range (31 bits of precision) and its real numbers have limited precision (about 16 decimal digits) and limited exponent size. Whenever Implemented Turing cannot carry out your program in the same way as Ideal Turing, it gives you a warning. It does not, however, warn you about the limited precision of real numbers. For example, if your program tries to create an array with a million elements in it, but there is not enough space, you will be warned.

Implemented Turing checks to make sure that your running program is meaningful. For example, it checks that variables are initialized before being used, that array subscripts are in bounds, and that pointers locate actual values. Checking guarantees *faithful execution*, which means that your program behaves as it would in *Ideal Turing*. If it does not, Turing gives a warning message. In production versions of Turing, checking can be turned off, for maximal efficiency of production programs.

Compile-Time Expressions

In certain situations, the Turing language requires you to use values that are known at compile time. Such values are called *compile-time values*. For example, the maximum length of a string type, such as the n in **string**(n), must be known at compile time. You are not allowed to read a value from the input and use it as the maximum length of a string. Here is a list of the places where *compile-time values* are required:

- (a) The maximum size n of a string, as in **string**(n).
- (b) The value of a label in a **case** statement or in a **union** type.
- (c) Each value used in **init** to initialize an array, record or union.
- (d) The lower bound L and upper bound U of a subrange, as in $L .. U$. There is, however, one exception to this rule. The exception, called a *dynamic array*, is an array declared using **var** or **const** that is not part of a record, union, or another array. In a dynamic array, the upper bound U (but not the lower bound L) is allowed to be read or computed at run time.

Case (d) implies that the size of **set** types is known at compile time. The technical definition of a *compile-time value* is any expression that consists of only:

- (1) explicit integer, real, boolean and string constants, such as 25, **false**, and "Charles DeGaulle", as well as enumerated values such as *color.red*.
- (2) Set constructors containing only compile-time values or **all**.
- (3) Named constants that name compile-time values.
- (4) Results of the integer operators +, -, *, **div** and **mod**.
- (5) Results of the *chr* and *ord* functions.
- (6) Results of the catenation operator +.
- (7) Parenthesized versions of any of these expressions.
- (8) OOT adds explicit character constants such as 'z' and 'Hello there'.

You are not allowed to use *run-time values* (any values not covered by items 1-7) in the places listed above (a-d).

Documentation of Language Features

The following syntax notation is used in this Manual:

- { *item* } means any number of repetitions of the *item* (or none at all)
- [*item*] means the *item* is optional

In the sections of the Manual that give an alphabetic list of language features, keywords, operators and delimiters, such as **get**, **procedure**, **:=** and **)** are written in

boldface. Comments in example Turing programs are written in *italics* in a smaller font than the rest of the program. Identifiers are written in *italics*. Explicit constants such as 27 or "Hello" are written normally.

In these sections, features are presented by giving their *syntax*, a general *description* of their meaning, *examples* of their use, and then *details* about the feature. The description is intended to satisfy the reader who mainly wants to know basic information about the item, for example, that the **string** type represents character strings. The examples illustrate important patterns of usage of the item, and should in many cases answer the question in mind. The detailed information that follows gives a full technical description of the item as well as references to related items. Virtually all the information in the Turing Report appears in this Manual, although not in its original form. This Manual takes liberties with the original syntax of the language to make explanations easier to understand. For example, the Manual describes *declarations* as a single item, and then explains restrictions that disallow certain declarations from appearing in particular contexts. By contrast, the Report uses the form of the context free syntax to imply, in a less obvious way, these same restrictions.

OOT Predefined Units

In 1994, changes were made in the way that OOT organizes predefined subprograms. Instead of a monolithic group of predefined subprograms, the subprograms are now organized into a number of different units. By default, all the predefined units are automatically made available to the user's program, but it is planned in future versions of OOT to allow users to select which units they wish to use in their program.

This Reference Manual lists the OOT predefined units in Chapter Four: Language Features, but it lists the subprograms in these units in Chapter Five: OOT Predefined Units. Subprograms in the predefined units are exported either qualified or unqualified.

Those subprograms that are exported unqualified do not require the unit name in the call (an example of this is the **round** function which is found in the **TypeConv** unit). These subprograms are also listed in the Language Features chapter of the Reference Manual. The subprograms exported qualified require the unit name in the call (an example of this is the **Create** subprogram in the **Dir** unit which must be called **Dir.Create**). Lastly, there are a number of subprograms that because implementation reasons are not actually placed in any predefined unit (an example of this is the **abs** function, that returns one of two types depending on the parameter type). However, each of these subprograms conceptually belongs to one of the units and is listed in the summary of that unit.

There is a separate unit called **Student** that contains all the predefined subprograms that are available in Original Turing (and are not exported by any other unit) and exports them all unqualified. This means that all Turing programs will still run perfectly under OOT. All the subprograms in the **Student** unit have counterparts in the rest of the OOT predefined units, which means that a program does not need the **Student** unit at all.

The predefined naming convention has been changed to take into account this new organization. The new convention is as follows:

Turing predefined subprograms (defined in Turing proper) are in lower case. Example: **drawbox**, **clock**.

OOT predefined subprograms that are exported unqualified are all in lower case. Example: **maxint**, **simutime**.

OOT predefined units have each the first letter of each word in the name capitalized. Example: **Draw**, **TypeConv**.

OOT predefined subprograms that are exported qualified have the first letter of each word in the name capitalized. Example: **Create**, **GetLong**.

OOT predefined constants that are exported qualified have the first letter in each word after the first word in the name capitalized. Example: **eBadStream**, **cdScreenWidth**.

Changes to OOT

Turing is still an evolving language. For the most up-to-date information on Turing, check Holt Software's web site located at <http://www.holtsoft.com/turing>.

Chapter 2

OOT for Windows

Installing OOT for Windows

Minimum System Requirements

Object Oriented Turing for Windows requires either:

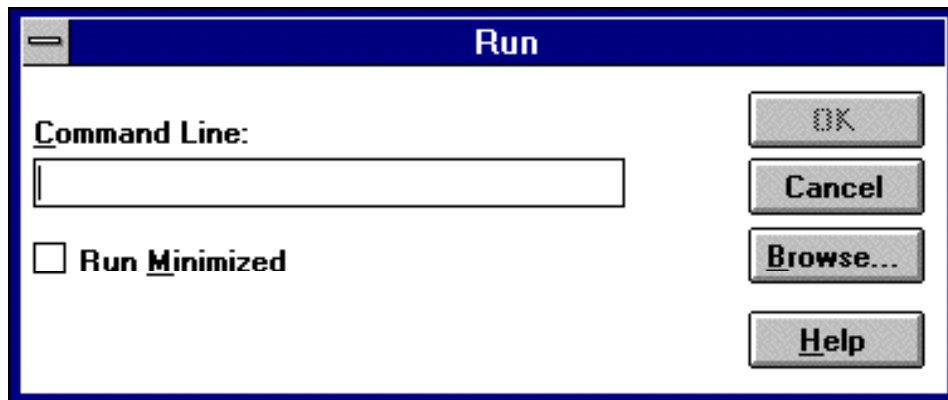
- (a) Microsoft Windows 95 or Windows NT
5 Mb of disk space
- (b) Microsoft Windows 3.1
386 or better processor
5 Mb of disk space
4 Mb of RAM and virtual memory turned on, or
8 Mb of RAM.

If you are uncertain as to whether your system is a 386, you can determine this by selecting **About Program Manager** from the **Help** menu in the **Program Manager** window. The last few lines of the dialog box that appears should read **386 Enhanced Mode** if your system is equipped with a 386 or better.

If you are uncertain as to the amount of memory on your system and whether you need to activate virtual memory, select **About Program Manager** from the **Help** menu in the **Program Manager** window. The second last line indicates the amount of memory available. If the line indicates that less than **4,000** KB of memory are free, you will need to enable virtual memory (detailed below).

Installing Under Windows 3.1

- 1) Start Microsoft Windows.
If at the DOS prompt, type **WIN** to start Windows.
- 2) Place the OOT for Windows Disk into drive A or B.
- 3) Select **Run** from the **File** menu in the **Program Manager**.



Run dialog box

- 4) A dialog box labelled **Run** appears. In it, there is an edit box labelled **Command Line:**. Enter into the box:

A:\SETUP

or

B:\SETUP

if the disk is in the B drive.

- 5) A dialog box appears labelled **Registration Information**. Please fill it out with your name and (if applicable) institution and press the **OK** button.
- 6) Another dialog box appears labelled **Registration Number**. Please fill in the edit box with your registration number.

Important: For the commercial version, your registration number is on a sticker labelled **OReg#** followed by 8 digits. The 8 digits are your registration number. The sticker will be in one of three places:

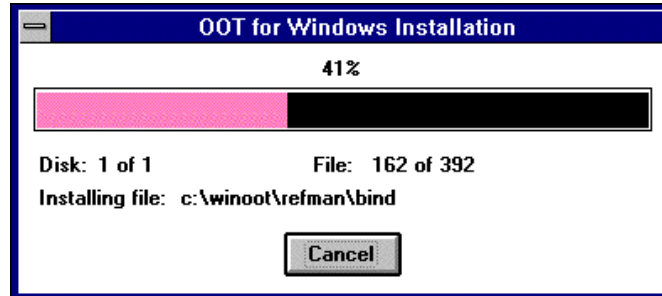
- 1) In the lower left hand corner on the first page of the *Turing Reference Manual*.
- 2) On the actual disk itself (if purchased without the *Turing Reference Manual*).
- 3) On the envelope that contained the software (if purchased with a textbook containing the software).

For the academic version, the registration number is found on the loose leaf release notes that accompanied the diskette. The academic registration number is 8 characters, an 'A' followed by 7 digits.



OOT for Windows Installation dialog box

- 7) Another dialog box appears labelled **OOT for Windows Installation**. The edit box indicates the directory into which the software will be installed. By default, this directory is **c:\winoot** which is suitable for most users. Press the **OK** button to accept the directory name.
- 8) The installation begins. A dialog box labelled **OOT for Windows Installation** appears. It contains a progress bar that indicates the amount of software already installed.



Progress Indicator window

- 9) If you are running on a system without a Math coprocessor (a 386 or a 486SX), when the installation is finished, the system will also install a driver in the **SYSTEM.INI** file and display a dialog box to that effect. This driver is used to perform math coprocessor emulation on systems without a math coprocessor. The added line reads:

device=c:\winoot\wemu387.386

De-installing Under Windows 3.1

- 1) Delete the icons - Select the WinOOT icon in the OOT for Windows program group and select **Delete** from the **File** menu. Click on the **Yes** button to delete the icon. Then select the OOT for Windows program group and once again select **Delete** from the **File** menu and click **Yes** to delete the program group.
- 2) Delete the driver reference - If you are running on a 386 or 486SX, the installer placed a driver in the **SYSTEM.INI** file. Using **Notepad** or any other text editor, open **c:\windows\system.ini**, edit out the line:
device=c:\winoot\wemu387.386
and save the result.
- 3) Delete the WinOOT files - Start the **File Manager** found in the **Main** program group. Select the **winoot** directory on the left- hand side. Select **Delete** from the **File** menu. A **Delete** dialog box comes up indicating the directory to delete. **Make certain this is the directory into which you installed the OOT files. Normally this should read "c:\winoot"**. Select **OK**. Another dialog box labelled **Confirm Directory Delete** appears. Check the directory name and then select **Yes to All**. A dialog box labelled **Confirm File Delete** appears. Click on the **Yes to All** button. All the WinOOT files are now deleted.

Enabling Virtual Memory Under Windows 3.1

- 1) Open the **Main** program group and double click on **386 Enhanced** icon.
- 2) A dialog box labelled **386 Enhanced** appears. Click on the button labelled **Virtual Memory** found on the right-hand side.
- 3) A dialog box labelled **Virtual Memory** appears. Click on the button labelled **Change>>** on the right-hand side.
- 4) The box expands. In the edit box labelled **New Size** found at the bottom, enter a figure of at least 4000. MS Windows will already have put a figure there depending on the amount of available disk space. If it is larger than 4000, feel free to use the amount indicated (although you should probably enter a figure no more than 8000). Press **OK** once a figure is entered.
- 5) A dialog box appears asking you to restart your machine. You should do so at this time.

Changing Monitor Resolution Under Win 3.1

Having installed WinOOT, you should now (if possible) set the resolution of the monitor to at least 800x600. It is not required, but it will certainly make the experience of using WinOOT somewhat easier as the windows won't overlap as much. Unfortunately, there isn't a single way of changing the resolution using MS Windows. We will describe the most common method, but you should consult your MS Windows and display card manuals before changing the monitor resolution.

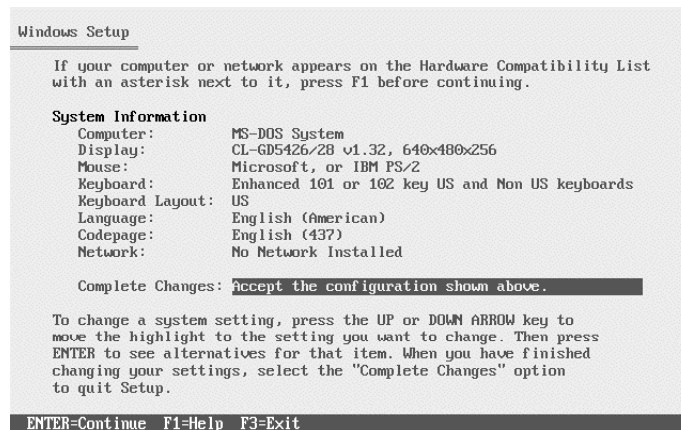
The most common method of changing the monitor resolution is to change the resolution from DOS. Enter the WINDOWS directory by typing:

```
cd \windows
```

Then run the **MS Windows SETUP** utility by typing:

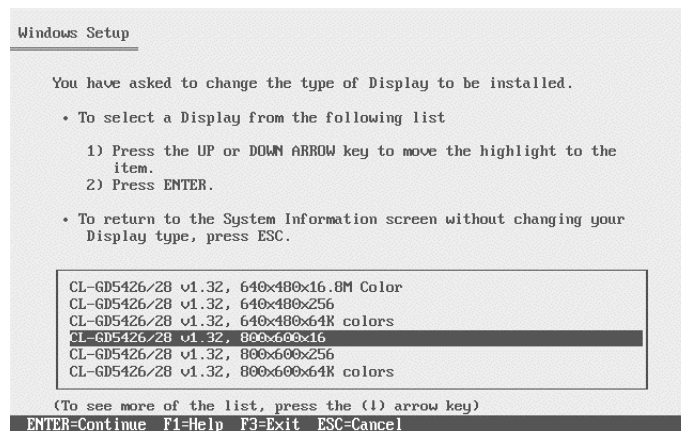
```
setup
```

This displays a list labelled *System Information*.



Windows Setup Screen

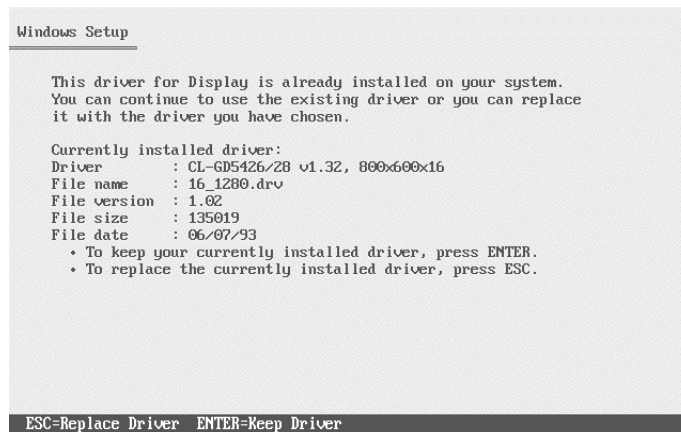
In this list there is a line titled *Display*. This line will normally list the display card and the resolution. Using the arrow keys, move the cursor up to this line and press the **ENTER** key. At that point a list of all the possible resolutions for the display card and monitor appears.



Windows Setup Monitor Selection Screen

Use the arrow keys to select a resolution (WinOOT functions best with at least 800x600x16 colors, although higher resolutions are fine) and press the **ENTER** key. This returns you to the **System Information** screen. Move the cursor to the bottom line, which says *Accept the configuration shown above* and press the **ENTER** key.

You then get another screen asking for confirmation of your choice.



Windows Setup Confirmation Screen

Pressing **ENTER** gives you one last screen of explanatory messages and you continue by pressing the **ENTER** key one last time. Finally you will be returned to DOS and are ready to start Windows.

Installing Under Windows 95

- 1) Place the OOT for Windows Disk into drive A or B.
- 2) Press the **Start** button and select **Run**.
- 3) A dialog box labelled **Run** appears. There is an edit box labelled **Command Line**. Enter into the box:

A:\SETUP

or **B:\SETUP** if the disk is in the B drive.

- 4) A dialog box appears labelled **Registration Information**. Please fill it out with your name and (if applicable) institution and press the **OK** button.
- 5) Another dialog box appears labelled **Registration Number**. Please fill in the edit box with your registration number.

Important: For the commercial version, your registration number is on a sticker labelled **OReg#** followed by 8 digits. The 8 digits are your registration number. The sticker will be in one of three places:

- 1) In the lower left hand corner on the first page of the *Turing Reference Manual*.

- 2) On the actual disk itself (if purchased without the *Turing Reference Manual*).
- 3) On the envelope that contained the software (if purchased with a textbook containing the software).

For the academic version, the registration number is found on the loose leaf release notes that accompanied the diskette. The academic registration number is 8 characters, an 'A' followed by 7 digits.

- 6) Another dialog box appears labelled **OOT for Windows Installation**. The edit box indicates the directory into which the software will be installed. By default, this directory is **c:\winoot** which is suitable for most users. Press the **OK** button to accept the directory name.
- 7) The installation begins. A dialog box labelled **OOT for Windows Installation** appears. It contains a progress bar indicating the directory into which the software will be installed. When it is completed, another dialog box appears stating that the installation is complete.
- 8) If you are running on a system without a Math coprocessor (a 386 or a 486SX), the system will also install a driver in the **system.ini** file. This is used to perform math coprocessor emulation on systems without a math coprocessor. The line reads

device=c:\winoot\wemu387.386

If you wish to remove WinOOT from your hard drive, you will need to eliminate this line from the **system.ini** file before deleting the directory containing the OOT files.

MS Windows Drivers for WinOOT

A note about the installed driver. In order to run, WinOOT installs a line calling a driver in the **system.ini** file in the **\windows** directory. The driver itself is placed in the directory containing the WinOOT files. If the WinOOT directory is ever deleted, it will be necessary to manually edit the **system.ini** file and remove the line calling the driver. Specifically, you must remove the line:

device=c:\winoot\wemu387.386

The directory specified in this line contains the WinOOT files. WinOOT uses this driver to emulate a 387 math coprocessor on systems that do not have one (the 386 and the 486SX).

As well, if WinOOT is ever accidentally installed twice, it is possible to end up with two copies of the **device=** line in your **system.ini** file. In this case, it will be necessary to edit out the second occurrence of the line. You will know if this occurs because you will not be able to enter MS Windows and will get a message indicating that there are two identical **device=** lines in the **system.ini** file.

Using OOT for Windows

Starting WinOOT

Once OOT is installed, you can launch the OOT for Windows application.

Windows 3.1

If the **OOT for Windows** program group is not already open, look for the program group icon labelled **OOT for Windows** and double click it. It opens and you see the contents of program group which contains a single icon also called **OOT for Windows**. Double click on this icon to startup WinOOT.

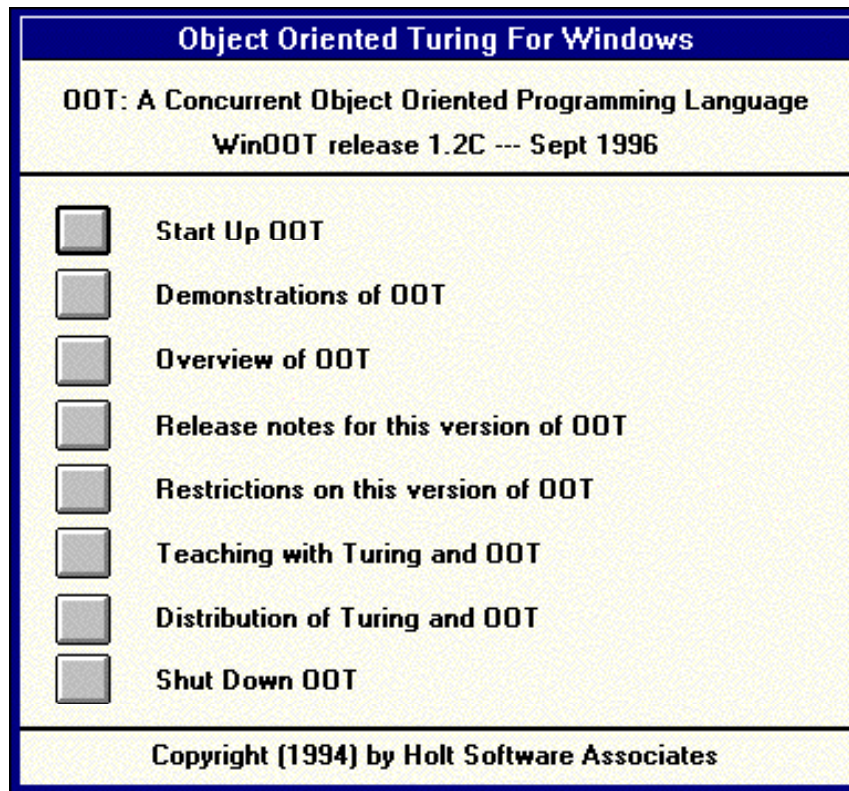
Windows 95

To start WinOOT in Windows 95, press the **Start** button and select **Programs** from the pop-up menu. From the *Programs* submenu, select **OOT for Windows**. From the **OOT for Windows** submenu, select the **OOT for Windows** program. Release the mouse to startup WinOOT.

Both

Once WinOOT is launched, you are presented with the initial startup dialog box which displays the version of OOT and the name of the person for whom the software is licensed. Please note that it is **not** legal to give the software accompanying this book to another person. The software is licensed for use by the purchaser of the book alone.

Pressing any key or clicking the mouse will advance to the next dialog box. This dialog box allows the user to start up the Turing environment or view OOT demonstration programs. You may wish to use these demonstration programs to familiarize yourself with the language.



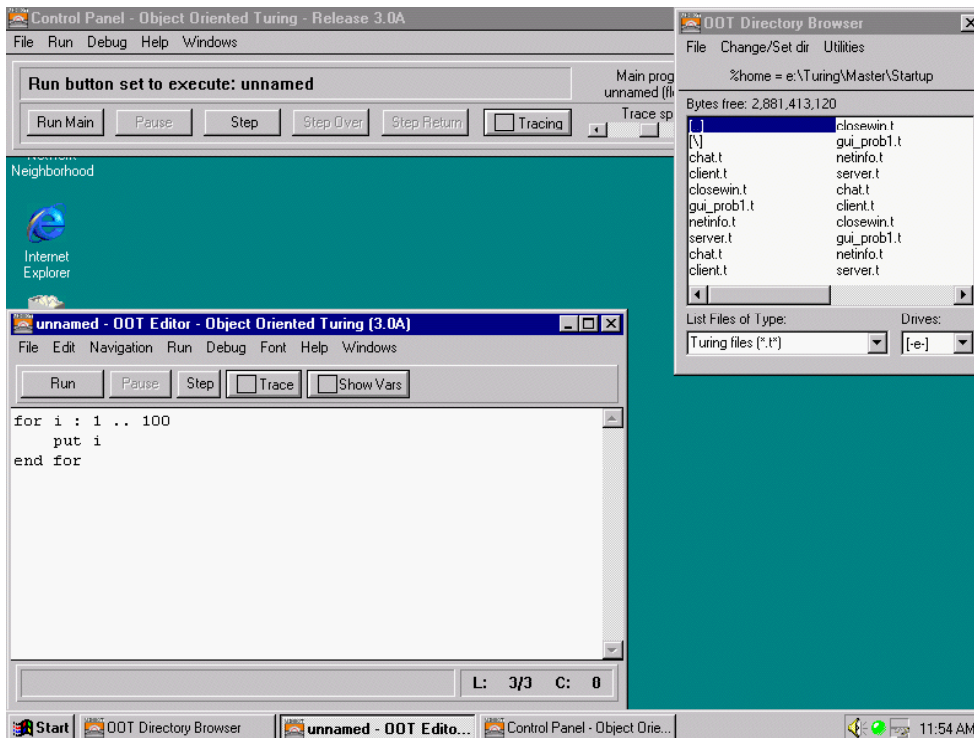
OOT Introduction Window

Besides **Start up OOT** and **Demonstrations of OOT**, you can select an overview of OOT, the current release notes for OOT, and other information, all of which will bring up text files with appropriate information relating to the button pressed.

Click on the **Start up OOT** button to start the OOT programming environment.

WinOOT Environment

When the WinOOT environment starts up, it creates three windows. In the upper-left corner, there is the **Control Panel**, in the upper-right corner of the screen there is the **Directory Viewer**, and in the lower-left corner, there is an **Editor window**. There are other types of windows that are used by WinOOT as well. These are the **Error Viewer**, **Debugger windows** and **Run windows**.



Screen Shot of WinOOT Running

OOT Windows - Control Panel



Control Panel

The Control Panel is a unique window (like the **Directory Viewer** and **Error Viewer**). The **OOT Control Panel** contains the menus for commands that, in general, do not involve changing the contents of a individual file. The commands found in the **Control Panel** are for controlling the environment as a whole, determining how programs are executed, and getting information about WinOOT and user programs. The **Control Panel** also contains a status message bar that is very useful. If you find yourself confused, look at the status

message bar of the **Control Panel**. It will tell you if you are currently executing a program, if there were errors encountered, and so on.

The **Control Panel** contains six parts. There is the window title bar, which also displays the version of OOT running. Note that you may be running a newer version than seen here as the software included with the book will be updated regularly. Beneath the title bar is the menu bar. The contents of the menu bar are documented in the section titled *Menus - Control Panel*. Beneath the menu bar is the status message bar, where all messages are displayed. Look at this area when you are unsure as to what OOT is doing. To the right of the status message bar, the name of the current *main program* is displayed. See *Running Programs - Programs with Multiple Parts* for more information about the *main program*.

Beneath the status message bar are a series of buttons. The **Run Main** button compiles and executes the *main program* (displayed above and to the right of the status message bar). When a program is executing, this button changes to **Stop** and stops program execution when pressed. The next button is the **Pause** button. This button pauses a program that is executing. When a program is paused, you can see the execution line or view variables (see *Stepping and Tracing* and *Showing Variables* for more information). When a program is paused, the **Pause** button changes to **Resume**. Pressing this button causes execution to continue from where it left off.

The **Step** button causes OOT to execute the next statement of a paused program (usually highlighted) and then pause, highlighting the line next line to be executed. If the statement to be executed is a call, an importation of a module, or is creating an object, OOT will step into the subprogram (or module or class initialization). This button is inactive when the program is executing and is not paused.

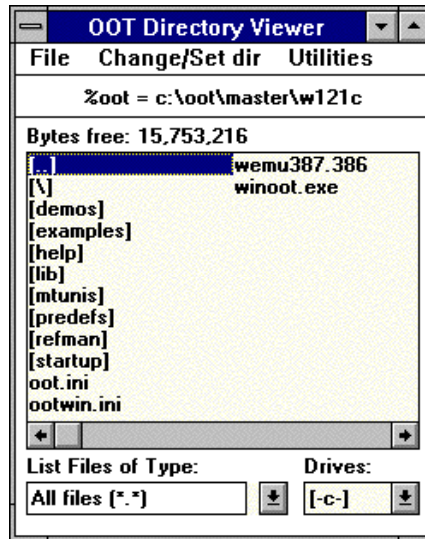
The **Step Over** button acts like **Step** except that if the current line is a call such as to a subprogram, pressing **Step Over** will cause the call to be made and execution to be paused at the line following the call.

The **Step Return** button operates like **Step** or **Step Over**, except that execution continues until the subprogram, module, or class initialization completes. The line highlighted will be the subprogram call. Pressing **Step** advances execution to the following line.

The **Tracing** button is a special button in that pressing it will toggle tracing on and off, as noted by the check box in the button. If tracing is on, whenever a program is executing the next line to be executed will be highlighted.

The speed of execution is controlled by the **Trace Speed slider** situated to the right of the buttons. This slider controls how fast the program executes while tracing is on. The more to the left the slider is placed, the longer OOT will pause on each line before executing the next line. When slid to the far right, OOT will execute the program (while displaying the next line to be executed) without pausing. See *Stepping and Tracing* for more details.

OOT Windows - Directory Viewer



Directory Viewer

The **Directory Viewer** is used to view the current directory and handle file operations. The top of the **Directory viewer** is the title bar. Beneath the title bar are the menus which are described in *Menus - Directory Viewer*. Just beneath the menus, the **Directory viewer** shows the name of the directory that is currently being displayed. Beneath the name, the amount of space free on the current disk is displayed. Below that is the main window which shows the files and directories in the current directory. You can use the menu commands to delete, rename, and copy these files, as well as to change the current directory

The main window displays the parent directory as [..]. Double clicking on it moves you up one directory level. The root (or top) directory is displayed as [\]. Double clicking on this symbol moves you to the root directory. Following the root directory, all subdirectories of this directory are listed in square brackets ([]). Double clicking on any of them moves you to the directory. Finally, all files in the directory are listed following the subdirectories. You can open any file into an **Editor window** by double clicking on the filename.

By default, the **Directory Viewer** displays all files. You can, however, specify the types of files you want displayed by selecting a different choice from the pop-up menu at the bottom left.

To change the directory being viewed, select the desired drive from the pop-up list of all available drives in the lower-right hand side.

By default, when you run a program, the directory displayed in the **Directory Viewer** is the execution directory. Files that are opened or created will be created relative to this directory. You can change this directory before you run your program, causing OOT to use a different directory to read and write data files. Note that file names specified using an

absolute path name do not use the execution directory when determining where the file is to be opened.

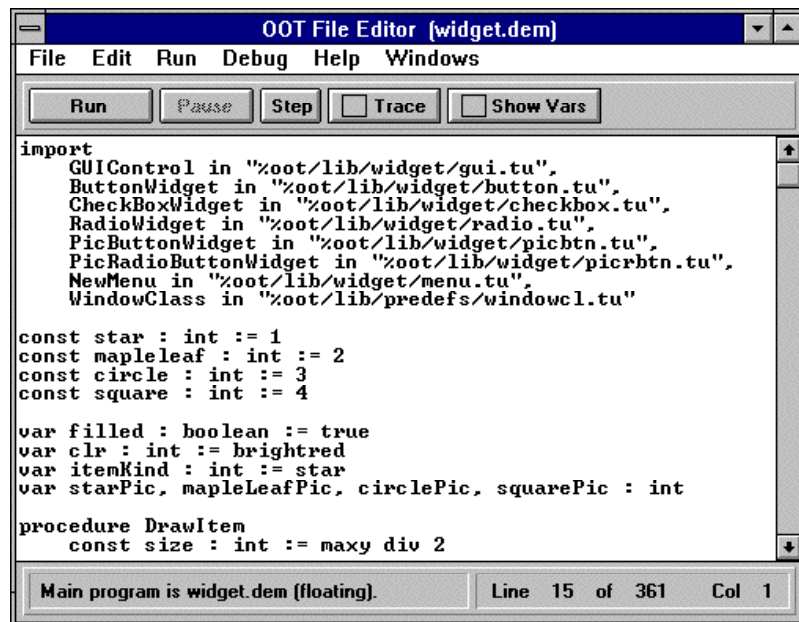
OOT Windows - Editor Window

An **Editor window** is used to enter and edit programs or to display text files.

The title bar of an **Editor window** contains the name of the file displayed in the window. If the file has not yet been named, it will appear as *unnamed*. Beneath the title bar is the menu bar. The menus contain the commands that allow you to edit the program and make changes to it. The contents of the menu bar are discussed in *Menus - Editor Window*.

Below the menu bar are five buttons used to control aspects of program execution as well as allowing the user to quickly close the file window. The **Run** button causes the program in the window to start execution. While the program is executing, the button changes to **Stop**. Pressing the **Stop** button immediately terminates execution of the program and causes the button to revert to **Run**.

The **Pause** button is used to temporarily halt execution of (pause) an executing program. While a program is paused, you can use the **Step**, **Step Over**, and **Step Return** buttons on the **Control Panel** or the **Step** button in the **Editor window** to step through execution of the program one line at a time. When there is no program executing, the **Pause** button is disabled. When a program is paused, the **Pause** button changes to **Resume** and pressing it resumes execution from where the program was halted.



Editor Window

The **Step** button is identical to the **Step** button in the **Control Panel**. While a program is stopped or paused, pressing the **Step** button causes OOT to execute one line of the program and highlight the next line to be executed.

The **Trace** is identical to the **Tracing** button in the **Control Panel**. This button selects whether tracing should occur while a program is being executed. When tracing is turned on, the program highlights each line as the line is executed. This slows down execution speed tremendously, but allows you to see how the program is being executed. Pressing the **Trace** button starts tracing either immediately (if the program is currently executing) or the next time the program is run. When tracing is turned on, the check box in the **Trace** button has an "X" in it. When the **Trace** button is pressed a second time, tracing is turned off and the "X" in the check box in the button is removed. You can control the speed at which the program execution is traced by using the **Trace Speed** slider in the **Control Panel**.

The **Show Vars** button is similar to the **Trace** button in that it toggles showing variables on and off. When selected, this button causes the currently executing program to display the list of units (if more than one), the global variables of the main unit, and the stack of the primary process. See *Showing Variables* for more information. Note that even when the **Show Vars** button is pressed again in order to turn this feature off, any still visible **Debugger windows** will be updated, slowing execution down. In order to stop the debugger from continuing to display variables, you must have turned **Show Vars** off and closed all open **Debugger windows**.

Beneath the buttons is the area displaying the text of the file. Text can go off the right and left edges of the screen. To move beyond the edges, select a line that extends beyond the edge of the screen and press the **End** key. To move off the left side of the screen (assuming that left-hand side of the screen is not column 1), press the **Home** key. This will move the text cursor to the beginning of the line and force the window to display column 1 on the left-hand side.

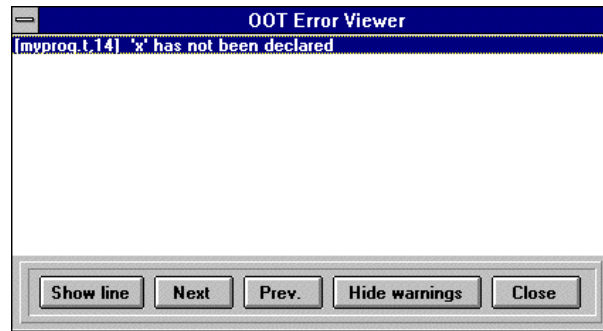
Beneath the text area is the status message bar. This displays any status messages and is identical to the status bar in the **Control Panel**. To the right of the status message bar is the cursor location information. It displays the line and column number of the text cursor and the total number of lines in the file.

OOT Windows - Error Viewer

This window lists any error messages that occur while compiling or executing the program. Note that there is no menu bar associated with the **Error Viewer**.

The body of the window displays all the error messages concerning the program. Each message starts with the file and line number that the error occurred on, followed by the text of the error. If the error extends beyond the right-hand side of the screen, a scroll bar will appear along the bottom of the error message area allowing you to view the entire error message.

Beneath the error message area are five buttons. The **Show line** button displays the line and (if possible) the token on the line that the error took place. If there are multiple errors in the program, the error currently highlighted in the **Error Viewer** is used to display the line.



Error Viewer

The **Next** button highlights the next error (wrapping around to the top error if necessary). It also displays the line the error occurred on and the token causing the error (if applicable). The **Prev** button highlights the previous error.

The **Hide warnings** button causes the **Error Viewer** not to display any warnings, only errors. This can be useful when certain constructs in the program are generating warnings that can be safely ignored. However, it is very hazardous to hide warnings as a matter of course. Warnings will often indicate a problem in the program that is not strictly a compilation error but may result in incorrect execution. By hiding the warning, the user does not get a chance to notice the problem and correct it before running the program.

The **Close** button closes the **Error Viewer**. The **Error Viewer** will reappear the next time an error occurs.

OOT Windows - Run Window



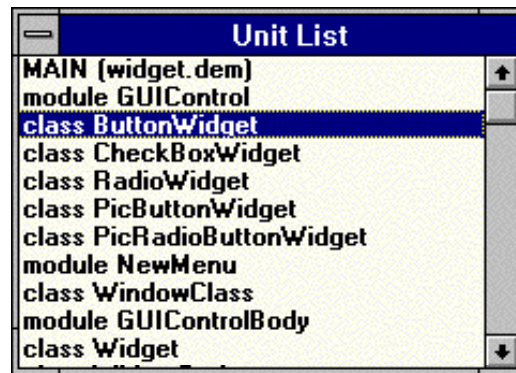
Run Window Displaying Output from %oot/examples/widget/widget.dem

All output from an OOT program appears in the **Run window**. In the example above, there is a large amount of graphical output appearing in the window. (Note that what appears as a menu bar at the top of the window is produced by the output of the OOT program being executed. It is *not* part of the **Run window** itself.)

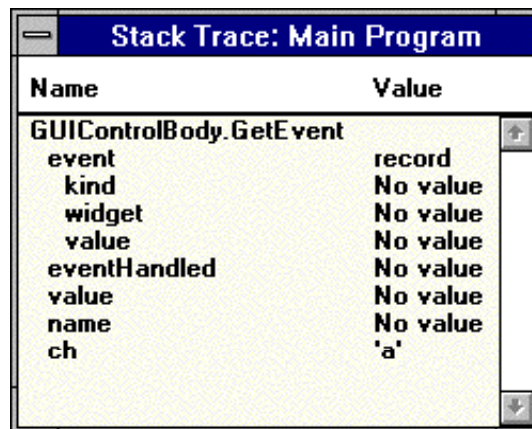
By default **Run windows** are 25 columns by 80 rows, although you can change this using the OOT Preferences. All output appears in the **Run window** except output being sent to files. It is also possible to have multiple **Run windows** open with output being sent to each one. The **Window.Open** in OOT can open another window.

To print the contents of a **Run window**, press Control+P when the window is selected. At present, you cannot save the contents of a **Run window**. Note that if you want to save the output of a run, you will need to redirect your output to a file. This is discussed in detail in *Running Programs - Input and Output Redirection*.

OOT Windows - Debugger Window



Debugger Window - Unit List



Debugger Window Showing Variable Contents

a[1..10] - D609D	
Showing: (1..10)	
Index	Values
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

Debugger Window Showing Array Contents

Whenever the user selects **Show Vars**, OOT displays the output in **Debugger windows**. There are three types of debugger windows. There is the **Unit List Debugger window**, which displays all the units found in the currently executing program. There are **Variable Display Debugger windows**, such as the second debugger window displayed. Lastly, there are **Array debugger windows**, such as the third window displayed. These display the contents of arrays. There are no menus associated with **Debugger windows**. For more information about viewing variables see *Showing Variables*.

Editing Text - Keyboard Editing Commands

There are a variety of keyboard commands that can be used in an **Editor window**. By learning these commands, you can edit your file more quickly and efficiently.

Moving the cursor:

Up Arrow Key	Move cursor up a line.
Down Arrow Key	Move cursor down a line.
Left Arrow Key	Move cursor left a character.
Right Arrow Key	Move cursor right a character.
Home key	Move cursor to the beginning of the current line.
End key	Move cursor to the end of the current line.
Page Up key (PgUp)	Move the cursor up a page.
Page Down key (PgDn)	Move the cursor down a page.
Ctrl+Left Arrow	Move the cursor left a word.
Ctrl+Right Arrow	Move the cursor right a word.
Ctrl+Home	Move the cursor to the top of the program.
Ctrl+End	Move the cursor to the bottom of the program.
Ctrl+Page Up, Ctrl+U	Move the cursor up half a page.
Ctrl+Page Dn, Ctrl+D	Move the cursor down half a page.
Ctrl+G	Go to specified line in the file (type the line number in the status bar).

Editing the text

Del	Delete character to right of cursor.
Backspace	Delete character to left of cursor.
Ctrl+J	Join the current line with following line.
Ctrl+E	Delete from the cursor to the end of the current line.
Ctrl+K	Delete the current line, place the cursor at beginning of following line.
Ctrl+Y	Insert a blank line before the current line, place the cursor at the beginning of the blank line.
Ctrl+L	Show search/replace dialog.
Ctrl+Shift %	Comment a selected range of lines.
Ctrl+5	Uncomment a selected range of lines.
Ctrl+Enter	Structure completion.
Ctrl+M	Structure match.
Ctrl+I	Paragraph program.

Editing Text - Selection, Cutting and Pasting

To move or duplicate large sections of text, *select* text, move it to the *paste buffer* (also called the *clipboard*) and then *paste* it into the text. Text can also be selected in order to perform other operations, such as commenting out or uncommenting a section of program, deleting a section, and so on.

Selection can be accomplished using the mouse or the keyboard. To select text using the mouse, click at the text position where the selection is to start. Holding the mouse button down, drag the mouse to the location in the text where the selection is to end. The selection grows from the starting position (called the *anchor point*) to the current mouse position. You can see the highlighted text easily as the selection is displayed in blue.

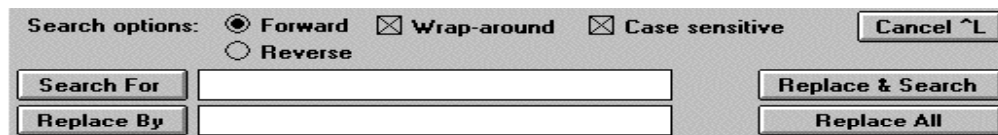
To select text using the keyboard, use the cursor keys to move the text cursor to where the selection is to begin. This location becomes the selection's anchor point. Holding the shift key down, use the arrow keys to move the text cursor around. The text between the anchor point and the current text cursor position is highlighted as the selection.

Once the text is selected, you can perform various operations on the selection. The simple operations are **Cut**, **Copy**, and **Paste**. **Cut** copies the selected text to the paste buffer and then deletes the text in the document. Use this when you want to move text to a different location in the document. **Copy** copies the selected text to the paste buffer but does not affect the location. **Paste** places the contents of the paste buffer at the text cursor location.

Note that using **Paste** does not delete the contents of the paste buffer. It is possible to paste the contents of the paste buffer several times without cutting or copying each time.

Editing Text - Searching and Replacing Text

When editing documents, it is often helpful to be able to either find or replace occurrences of one string for another in a file. This capability is called *search and replace*.



Search/Replace Dialog Box

To display the Search/Replace dialog, select **Show search/replace** from the **Edit** menu in an **Editor window** (or press Control+L). The bottom of the **Editor window** displays the Search/Replace dialog. To make the dialog box disappear, select **Hide search/replace** from the **Edit** menu (the **Show search/replace** menu item becomes **Hide search/replace** when the Search/Replace dialog is visible), press Control+L or press the **Cancel** button in the Search/Replace dialog.

There are several sections to the Search/Replace dialog. The most important parts are the two text boxes in the middle called the *search field* and the *replace field*. The user is expected to enter the text that is to be found in the *search field* and enter the text that is to replace the found text in the *replace field*. The contents of the *search field* is called the *search text*. The contents of the *replace field* is called the *replace text*. When a search is done, OOT scans the program text looking for occurrences of the *search text*. When the replace command is given, the found text is replaced with the *replace text*.

In the top right of the Search/Replace dialog there are a pair of radio buttons labelled **Forward** and **Reverse**. When **Forward** is selected, OOT searches forward (downward) through the program searching for occurrences of the *search text*. When **Reverse** is selected, OOT searches backwards (upward) through the program searching for occurrences of the *search text*. To the right of the radio buttons is a check box labelled **Wrap-around**. When this box is checked, whenever a forward search reaches the bottom of a file, the search is continued from the top (although it stops if it searches the entire file and does not find anything). Likewise, if **Wrap-around** is selected, any backwards search that hits the top of the file continues searching from the very bottom.

The check box to the right of **Wrap-around** is **Case sensitive**. When **Case sensitive** is selected, searches only match text in the program that is identical to the *search text*. When **Case sensitive** is not selected, searches will match text in the program that matches the *search text*, disregarding the case of both.

There are five buttons in the Search/Replace dialog. The **Search For** button immediately searches either forward or backward for the next occurrence of the *search text* in the program, scrolls to the location in the **Editor window**, and highlights the matching selection. The **Replace By** button replaces the selection found using the **Search For** button with the *replace text*.

The **Replace and Search** button replaces the currently found text with the contents of the *replace text* and then immediately searches for the next (or previous) occurrence of the *search text*.

The **Replace All** button goes through the file in the Editor window from top to bottom and replaces all occurrences of the *search text* with the *replace text*.

Lastly, the **Cancel** button causes the Search/Replace dialog to disappear.

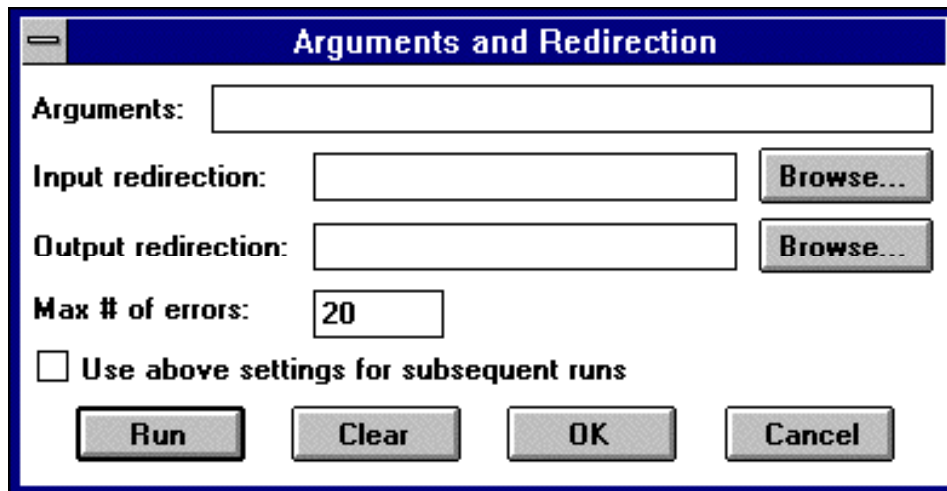
A search and replace is usually done by entering text in both the Search field and the Replace field, doing a **Search For** the text and then pressing either the **Replace and Search** button to replace a found occurrence of the contents of the Search field or alternatively pressing **Search For** in order to skip over an occurrence. In both cases, the next occurrence of the contents of the Search field is highlighted. This continues until the entire file has been checked.

Running Programs - Input and Output Redirection

To run a program, press the **Run** button in the **Control Panel** or **Editor window**, or select the **Run** menu item from the **Run** menu in either type of window. You can also press **F1** when an **Editor window** or the **Control Panel** is selected. The Run command compiles and executes the selected window (if the currently selected window is an **Editor window**), the last selected **Editor window** (if the currently selected window is the **Control Panel**), or a fixed selected program. See *Running Programs - Programs with Multiple Parts* for more details on how to select the file to be run.

Once the program is running, a **Run window** will be created and the output of the program displayed in the **Run window**. You cannot save the contents of a **Run window**, but you can print it by pressing Ctrl+P on the keyboard while the **Run window** is active.

You can send all the text output of a program to a printer by redirecting the output of the program to a file using the **Run with Arguments** command, opening the text file produced in an **Editor window**, and printing the file by selecting **Print** from the **File** menu.



Arguments and Redirection Dialog Box

You can also specify that input should come from a file instead of the keyboard. Both these operations are performed by selecting **Run with Args** from the **Run** menu. When you do so, the **Arguments and Redirection** dialog box appears.

This dialog box has several parts. The first part is the **Arguments** field. You can enter any text in the **Arguments** field and the text will be passed as command line arguments to the program when it executes. The program can read the command line arguments using the OOT predefined subprograms **nargs** and **fetcharg**.

In the **Input redirection** field, you can either type in a file name or press the **Browse** button to the right to bring up a file dialog which allows you to select the file that should be used. If this field is filled in, when the program executes, any input that would normally

have been read from the keyboard (usually using **get**) will be read from the specified file instead. Note that the input read from a file will not be echoed in the **Run window**.

In the **Output redirection** field, as in the **Input redirection** field, you can either enter a file name into the field or press the **Browse** button beside the field to select a file name to be used for output redirection. When this field is filled in, any text output (usually using **put**) will be sent to the specified file instead. The file produced will be a text file and can be opened as an **Editor window**. Note that if no output is to be sent to the screen at all, OOT will not display a **Run window**. Note also that graphical output will still be sent to the **Run window** and not sent to the file.

The **Output redirection** field shows the maximum number of errors that you wish to have returned at once from the compiler.

Beneath the **Maximum Errors** field is a check box that sets whether all subsequent runs should run with the arguments set in this dialog box. If set, the **Run** button in the **Editor windows** and **Control Panel** change to say **Run w/ Args** to remind you that the input and output may be redirected.

Beneath the **Maximum Errors** field are four buttons. The **Run** button runs the program with the information that you have entered here. The **Clear** button wipes out all the information in the dialog box. The dialog box always stores the information about the last **Run with Arguments**. If you want to change all the values quickly, use the **Clear** button. The **OK** button is used to set the **Arguments and Redirection** without actually running the program. The **Cancel** button aborts the run without changing the contents of the dialog box.

Besides the **Run** and **Run with Arguments** menu items, the other useful menu items in the **Run** menu are the **Set preprocessor symbols**, to set preprocessor symbols for **#if** statements in your program, and the **Compiler reset** menu item which resets the compiler, forcing a recompilation of all the units. This is useful if the environment shows odd behavior.

In future versions of the software, there may be a **Compile** menu item in the **Run** menu. This will be used for producing standalone versions of compiled programs (essentially EXE files that you can run without having OOT present). When this becomes available, an item in will appear in the **Help** menu explaining the use of this feature.

Running Programs - Programs with Multiple Parts

In OOT, editing and executing a program consisting of a single file (no **import** or **include** statements) is straightforward. When the **Run** or **Run with Arguments** command is given, OOT compiles and executes the contents of the active **Editor window**, or if no **Editor window** is active, the last active **Editor window**. The program that is compiled and executed when the run command is given is called the *main program* and its name is displayed in the **Control Panel**.

In cases where a user is editing and executing programs consisting of several units (usually a main program and several modules or classes in separate files), it is not always obvious which file should be run. Running the currently active **Editor window** will not work if the user is editing an **Editor window** containing one of the classes the program uses instead of the main program itself.

Rather than always having to activate the window containing the main program before selecting **Run**, OOT provides an alternative method of specifying which program to compile when the run command is selected.

This is done by setting the *main program* to be either *fixed* or *floating*. When the *main program* is *floating* (the default method), the *main program* is either the currently active **Editor window**, or if no **Editor window** is active, the last active **Editor window**. If the *main program* is *fixed* then the *main program* is a specific file that the user previously chose.

For example, a file called *database.t* uses two units *readdata.tu* and *writedata.tu*. The user would specify *database.t* as the *main program*. Then regardless of which file is being edited, when the run command is given, OOT compiles and executes *database.t*.

If the user later wants to run a different program, she or he sets the *main program* to be *floating*. OOT then compiles and executes the active **Editor window**.

The following commands relate to selecting the *main program*. The **Run** button in the **Control Panel** and in **Editor windows** compiles and runs the *main program*. The current *main program* along with whether it is *fixed* or *floating* is found to the right of the status message bar in the **Control Panel**. See the section labelled **Control Panel** for a picture.

The **Show main program** menu item in the **File** menu of the **Control Panel** activates the *main program*, opening it into an **Editor window**, if it is not already open.

The **Run** and **Run with Arguments** commands in the **Control Panel** and the **Editor windows** compile and execute the *main program*.

The **Fix main program** menu item in the **Change/set dir** menu in the **Directory viewer** fixes the *main program* to be the file currently selected in the **Directory viewer**. Using this command means that the *main program* need not be opened to execute it, a convenience when only the units of a program need modification.

The **Fix main program** menu item in the **File** menu in the **Editor window** fixes the *main program* to be the contents of active **Editor window**.

The **Float main program** in both the **Directory Viewer** and the **Editor windows** causes the *main program* to *float* once again.

Menu Commands - Control Panel

File

New: Open an empty untitled **Editor window**.

Open: Bring up the **Open** dialog box to allow the user to open a file into an **Editor window**.

Show main program: Display the main program window. If hidden, bring it to the front and make it active.

Show directory viewer: Display the **Directory Viewer**. If hidden, bring it to the front and make it active.

OOT Preferences: Bring up the OOT Preference dialog box to allow the user to change a variety of features in OOT such as the default size of **Editor windows** and **Run windows**, the cursor shape, and so on.

Quit OOT: Quit the OOT environment. Close all OOT windows. Ask the user whether she or he wishes to save the contents of **Editor windows** that have been modified.

Run

Run main: Run the main program. The main program is specified in the **Control Panel** to the right of the status message bar.

Run with arguments: Bring up the **Arguments and Redirection** dialog box. See *Running Programs*.

Single Step: Execute a single line of the program and highlight the next line of the program to be executed. See *Stepping and Tracing*.

Pause Execution: Pause execution of the program. While the program is paused, the user can step through execution and view variables. This menu item becomes **Resume Execution** when the program is paused.

Stop Execution: Terminate execution of the program.

Stop/Close Run Window: Terminate execution of the program and close all the **Run windows**.

Set Preprocessor Symbols: Bring up a dialog box allowing the user to specify all the preprocessor symbols for use with the **#if** statements in an OOT program.

Compiler Reset: Reset the compiler. Throw away all compiled units.

Debug

Show unit list: Display the list of all units (main unit, modules, and classes) used by the currently executing program. See *Show Variables*.

Show processes: Display the list of all processes used by the currently executing program. See *Show Variables*.

Show queues: Display the list of all processes used by the currently executing program by the queue that they are in (the queues can be *executing*, *waiting for event*, *paused*, and so on).

Trace on/off: Turn tracing on or off. Same as pressing the **Trace** button in the **Control Panel**. See *Stepping and Tracing*.

Show vars on/off: Turn Show Vars on or off. Same as pressing the **Show Vars** button in the **Control Panel**. See *Show Variables*.

Show allocated objects: Show some objects allocated by the currently executed OOT program. These objects will include file stream numbers, directory stream numbers, Pics, and other elements from the predefined modules in later versions of the OOT software.

Show run errors: Display the Run Error Viewer. When this box is visible, any non-fatal run time errors (such as file opening failure, illegal font numbers, etc.) are displayed here. Use this when predefined subprograms do not perform as expected. Note that it is better to check and handle possible failures in the program itself.

Show memory usage: Display a dialog box containing the total amount of free memory, the largest contiguous block of memory and the percentage of Windows resources in use. If OOT is running out of memory, select this item before running the program. You should have a minimum of 1 megabyte of free memory before the program executes.

Show execution line: Highlight the line that is currently being executed. Useful for finding out what section of the program is currently being run.

Show definition: Display the definition of the highlighted variable. This can only be used when a **Debugger window** was previously selected. The source declaration of the variable being highlighted in the **Debugger window** will be highlighted.

Help

OOT Reference: A submenu containing help files about the help system, the OOT environment, the editor, the debugger, and other parts of the environment.

About: Information about the current version of OOT for Windows and how to contact Holt Software.

Authors: The list of those who have contributed to the OOT for Windows software.

Versions: Versions of different parts of the OOT environment.

Information: Display dynamic information about the OOT environment in the status message bar. Information items include the name of the main program, the current run time arguments, the current preprocessor symbols, the current input file, the current output file, the execution directory, and the current directory.

Windows

This menu contains the list of all open windows in OOT. Non-unique windows are found in submenus labelled **Editor Windows**, **Run Windows** and **Debug Windows**. Selecting a menu item from this menu causes the corresponding window to come to the front and to become active.

Menu Commands - Directory Viewer

File

New: Open an empty untitled **Editor window**.

Open: Bring up the **Open** dialog box to allow the user to open a file into an **Editor window**.

Refresh: Refresh the **Directory Viewer**, displaying any new files and removing any older files.

Run current main: Run the current main file.

Stop Execution: Terminate execution of the program.

Pause/Resume execution: Pause execution of the program. While the program is paused, the user can step through execution and view variables. If the program is already paused, then resume execution.

Select all files: Select all files in the directory being displayed by the **Directory Viewer**.

Quit OOT: Quit the OOT environment. Close all OOT windows. Ask the user whether she or he wishes to save the contents of **Editor windows** that have been modified.

Close Viewer: Close the directory viewer. Selecting **Directory Viewer** menu item from the **Window** menu causes the **Directory Viewer** to reappear.

Change/Set dir

OOT [%oot]: Move to the OOT directory. By default, this directory is the directory where the OOT for Windows executable is located.

Job [%job]: Move to the job directory. The job directory is selected by user using the **Set job directory** menu item in this menu. It is assumed that the user would set the job directory to the directory containing most of the files for the project that he or she is working on at the moment.

Home [%home]: Move to the user's home directory. By default, this is the directory that OOT starts up in. The startup directory is specified in the **Working Directory** field of the **Program Item Properties** dialog box for *Object Oriented Turing for Windows*. This dialog box is modified from within the Windows Program Manager, not within OOT.

Help [%help]: Move to the OOT Help directory. Various help text files are stored here.

Temp [%tmp]: Move to the directory specified by the TMP environment variable in DOS. This directory is also used by Microsoft Windows for various temporary files.

Other...: Move to the directory specified in the dialog box that appears.

Fix main program: Set the selected file to be the main program. A single file must be selected in the **Directory Viewer**. See *Running Programs - Programs with Multiple Parts*.

Float main program: Set the main file floating again. See *Running Programs - Programs with Multiple Parts*.

Fix exec. directory: Set the current directory to be the execution directory. If this is set, then all execution takes place in this directory rather than the current directory.

Float exec directory: Set the execution directory floating again.

Set OOT [%oot] directory: Set the current directory to be the OOT directory. Note that many example programs use the %oot directory as the way of locating system files.

Set home [%home] directory: Set the current directory to be the home directory.

Set job [%job] directory: Set the current directory to be the job directory.

Utilities

Delete files [del/rm]: Delete any selected files.

Rename file/dir [ren]: Rename the selected file or directory. A dialog box comes up allowing the user to specify the new name.

Move files [move]: Move a set of files to a new directory by allowing the user to specify the location of the directory to move the files to in a dialog box.

Copy files [copy/cp]: Copy a set of files to a new directory by allowing the user to specify the location of the directory to copy the files to in a dialog box.

Backup files: Make a copy of any selected files with the **.BAK** extension.

Delete directory [del+rmdir]: Delete the selected directory and its contents.

Create directory [mkdir]: Create an empty directory with the name specified in the dialog box that appears.

File pattern search [grep]: Search the contents of a set of files for a specified pattern.

Add to OOT arguments: Add the currently selected files to the OOT run time arguments list. See *Running Programs* for more details.

Set as OOT input file: Select the currently selected file to be the OOT input redirection file. See *Running Programs* for more details.

Set as OOT output file: Select the currently selected file to be the OOT output redirection file. See *Running Programs* for more details.

Menu Commands - Editor Window

File

New: Open an empty untitled **Editor window**.

Open: Bring up the **Open** dialog box to allow the user to open a file into an **Editor window**.

Close: Close this **Editor window**. If the window has been changed since it was last saved, there will be an opportunity to save it.

Save: Save the contents of the **Editor window** in the file associated with the window. If the window is unnamed, bring up a **Save As** dialog box to allow the user to select the name and directory of the file to be saved.

Save As: Bring up the **Save As** dialog box. Save the contents of the **Editor window** in the file selected. Note that this changes the file associated with the window to be the new file.

Revert: Discard any changes since the program was last saved.

Print: Print the program on the printer selected in Microsoft Windows.

Navigation: A submenu of commands to change the text cursor position. Mostly used to remind users of the keyboard shortcuts.

Fix main program: Make this the main program. Whenever a the user selects **Run**, either from menu, button or keyboard shortcut, this will be the program that is run until another file is made the main program or **Float main program** is selected. See *Running Programs - Programs with Multiple Parts*.

Float main program: Cause the main program to float again. When the main program is floating, the active or last active **Editor window** will be the one that is run when the run command is given. See *Running Programs - Programs with Multiple Parts*.

Redraw: Redraw the contents of the **Editor window**. Select this command if for some reason the appearance of the contents of the **Editor window** gets scrambled.

OOT Preferences: Bring up the OOT Preference dialog box which allows the user to change a variety of features in OOT (such as the default size of **Editor windows** and **Run windows**, the cursor shape, and so on).

Quit OOT: Quit the OOT environment. Close all OOT windows. Ask the user whether she or he wishes to save the contents of **Editor windows** that have been modified.

Edit

Undo: Undo the last operation, restoring the contents of the window to what they were before the operation.

Cut: Copy the selection to the paste buffer and then delete the selection. See *Selection, Cutting and Pasting* for more details.

Copy: Copy the selection to the paste buffer. See *Selection, Cutting and Pasting* for more details.

Paste: Paste the contents of the paste buffer at the text cursor. If there is a selection, the paste buffer replaces the selection. See *Selection, Cutting and Pasting* for more details.

Join Lines: Join the line that the text cursor is on with the following line, adding a space between the two lines.

Delete to EOL: Delete from the text cursor to the end of the line that the text cursor is on.

Delete line: Delete the entire line that the text cursor is on.

Open new line: Insert a new empty line on the line before the line that the text cursor is on and place the text cursor at the beginning of this new line.

Show search/replace: Show the **Search/Replace** dialog box at the end of the window. If the dialog box is already visible, this menu item becomes **Hide Search/Replace** and selecting it hides the **Search/Replace** dialog box. See *Searching and Replacing Text* for more details.

Comment range: “Comment out” all lines in the selection. It places “%-” at the beginning of all lines in the selection. Any line starting with a “%” is treated as a comment by the compiler and ignored. This is useful for temporarily disabling sections of the program while testing it.

Uncomment range: “Uncomment” all lines in the selection. Removes the “%-” at the beginning of any lines in the selection. This command is used in conjunction with the **Comment range** command to reenable sections of the program that were disabled using the **Comment range** command.

Structure completion: If the text cursor is at an OOT program construct (for example **loop**, **case**, and so on.), this command places the end-construct after the construct, adds a blank line between the two, and moves the text cursor there, indenting the line by four spaces. For example, if you have:

```
loop^
```

where ^ represents the text cursor, selecting **Structure completion** produces the following:

```
loop
  ^
end loop
```

Structure match: If the text cursor is at an OOT program construct (for example **loop**, **case**, etc.), or at an end-construct (for example **end loop**, **end case**, etc.), this command moves the text cursor to the end of that particular construct. If at the end of the construct, **Structure match** moves to the beginning of the construct. For example, if you have:

```
for i : 1..20^
  loop
    loop
      put "A"
    end loop
  end loop
end for
```

where ^ represents the text cursor, selecting **Structure match** moves the cursor to the beginning of the **end for**:

```
for i : 1..20
  loop
    loop
      put "A"
    end loop
  end loop
^end for
```

selecting **Structure match** a second time moves the cursor to the beginning of the first line.

Paragraph file: Indent the file, splitting lines that are too long. This command formats the file, indenting constructs properly and splitting lines that are longer than 78 characters or have more than one statement on them (if possible). For example, if you have:

```
for i : 1..20
loop
if x = 3 then put "A" end if
end loop
end for
```

selecting **Paragraph file** produces the following:

```
for i : 1..20
  loop
    if x = 3 then
      put "A"
    end if
  end loop
end for
```

Run

Run: Run the main program. If the main program is floating, run this **Editor window**. You can always determine the current main program by looking to the right of the status message bar in the **Control Panel**.

Run with Args: Bring up the **Arguments and Redirection** dialog box. See *Running Programs - Input and Output Redirection*.

Pause execution: Pause execution of the program. While the program is paused, the user can step through execution and view variables. This menu item becomes **Resume Execution** when the program is paused.

Single Step: Execute a single line of code and highlight the next line of code to be executed. See *Stepping and Tracing*.

Stop execution: Terminate execution of the program.

Set preprocessor symbols: Bring up a dialog box allowing the user to specify all the preprocessor symbols for use with the **#if** statements in an OOT program.

Compiler reset: Reset the compiler. Throw away all compiled units.

Debug

Next error: Highlight the next error in the list of errors. The window where the error lies will be brought to the front and activated. The line where the error occurred

will be highlighted and, if possible, the token causing the error will be marked. If at the end of the list, wrap around and highlight the first error.

Previous error: Highlight the previous error in the list of errors. If at the beginning of the list, wrap around and highlight the last error.

Show unit list: Display the list of all units (main unit, modules, and classes) used by the currently executing program. See *Show Variables*.

Show processes: Display the list of all processes used by the currently executing program. See *Show Variables*.

Show queues: Display the list of all processes used by the currently executing program by the queue that they are in (the queues can be *executing*, *waiting for event*, *paused*, and so on.).

Show allocated objects: Show some objects allocated by the currently executed OOT program. These objects will include file stream numbers, directory stream numbers, Pics, and other elements from the predefined modules in later versions of the OOT software.

Show run errors: Display the Run Error Viewer. When this box is visible, any non-fatal run time errors (such as file opening failure, illegal font numbers, etc.) are displayed here. Use this when predefined subprograms do not perform as expected. Note that it is better to check and handle possible failures in the program itself.

Show execution line: Highlight the line that is currently being executed. Useful for finding out what section of the program is currently being run.

Trace on/off: Turn tracing on or off. Same as pressing the **Trace** button in the **Control Panel**. See *Stepping and Tracing*.

Show vars on/off: Turn Show Vars on or off. Same as pressing the **Show Vars** button in the **Control Panel**. See *Show Variables*.

Help

Help with Editor: Display a text file containing help on the editor. In later versions of OOT, if there are commands in the editor that are not documented here, check the Editor help file using this command for instructions on use.

Keyword usage: If the text cursor is at an OOT predefined subprogram or part of the OOT language, this command displays a one line summary of the OOT Reference Manual entry in the status bar.

Keyword reference: If the text cursor is at an OOT predefined subprogram or part of the OOT language, this command displays a window containing the OOT Reference Manual entry on the item.

OOT Reference: A submenu containing help files about the help system, the OOT environment, the editor, the debugger, and other parts of the environment.

About: Information about the current version of OOT for Windows and how to contact Holt Software.

Authors: The list of those who have contributed to the OOT for Windows software.

Versions: Versions of different parts of the OOT environment.

Information: Display dynamic information about the OOT environment in the status message bar. Information items include the name of the main program and information from the arguments and redirection dialog box.

Windows

This menu contains the list of all open windows in OOT. Non-unique windows are found in submenus labelled **Editor Windows**, **Run Windows** and **Debug Windows**. Selecting a menu item from this menu causes the corresponding window to come to the front and to become active.

Debugging - Stepping and Tracing

It is very useful to be able to follow the execution of a program visually. OOT lets you see each line of a program as it is executed. By doing so, you can see exactly how the program *is* executing, compare it with how it *should be* executing, and discover bugs.

As well, viewing the execution of the program can be an instructional aid to learning how various language constructs operate.

Stepping and tracing provide two ways of seeing a program execute. Stepping is when OOT executes a small chunk (usually one line) of a program and then waits for another command. Tracing is when a line being executed is highlighted for a set amount of time, then the next line, and so forth. In Tracing, execution continues until the user gives a command to stop execution (or the program terminates). In stepping, the user must give a command to continue after each line (or chunk) or code.

Tracing

To start OOT tracing a program, press the **Trace** button in an **Editor window** (causing the check box in the button to be filled). If a program is already executing, OOT starts highlighting the line being currently executed and continues to execute the program. Otherwise, OOT starts tracing execution the next time a program is run. To turn off tracing, press the **Trace** button again. (Note that the check box becomes empty.)

You can control the speed at which tracing occurs by adjusting the slider labelled **Trace Speed** on the right-hand side of the **Control Panel**. To have each line executed more quickly, move the slider to the right. To slow execution down, move it to the left. Anytime you want execution to resume normally, click the **Trace** button again. You can go back and forth turning tracing on and off as you deem appropriate.

Stepping

You can also examine execution one line at a time (i.e. stepping). While the program is running, click on the **Pause** button in the **Editor window** (note that it changes to read **Resume**). Once the program is paused, the three buttons in the **Control Panel** labelled

Step, **Step Into**, and **Step Return** are enabled. Clicking the **Step** button causes OOT to execute one line of code (the next line will be highlighted). If you are at a call to a subprogram, the **Step** button enters the subprogram and highlights the first line of it. If you wish to trace execution over the subprogram (in other words, execute the subprogram all at once and pause at the statement after the call), click on the **Step Over** button. As well, if you are in a subprogram and wish to quickly skip the rest of a subprogram, click the **Step Return** button, which executes the rest of the procedure and pauses execution at the statement following the call to the subprogram.

Note that pressing **Step Return** in the main program is the same as pressing the **Step** button.

Note that if you are stepping over multiple lines (by doing a **Step Over** or a **Step Return**) while tracing is turned on, OOT will not highlight the individual lines being executed.

When stepping through a program, you can always continue full speed execution by making certain that tracing is turned off and pressing the **Resume** button.

Breakpoints

OOT does not have the ability to dynamically set breakpoints (points where execution of the program stops and allows the user to start stepping or tracing). However, the user can set a breakpoint in the program. This is done by placing the **break** statement anywhere in the program. When the program reaches the **break** statement, execution is paused just as if the user pressed the **Pause** button at that point. The user can then issue any of the stepping, tracing or view variable commands, or press the **Resume** button to continue execution. The user can have as many **break** statements in the program as he or she wishes. Execution will pause at each **break** statement.

Debugging - Showing Variables

OOT is capable of showing all the variables in a program. To understand how OOT displays the variables, you must understand the way the variables are grouped. First there are unit variables, grouped one window per unit, and then there are stack variables, grouped one window per process. A program consisting of one unit (no **import** statements) and one process (no concurrency) would have one window of unit variables and one window of stack variables. When observed in OOT, the two debugging windows would be considered the "global" variables and the "local" variables.

A unit is either the main program or it is a module or class in a separate file with the keyword **unit** at the top of the file. OOT programs use units using the **import** statement. Unit variables are those in a given unit that are declared outside of any subprograms. These variables are common to all subprograms in a unit. In the main program unit, they are known as global variables.

A process is one thread of execution. Each process has its own local variables. The stack variables for a particular process consist of all the local variables in the subprograms that the process is currently in. In simple terms, stack variables are called local variables.

Displaying Variables

To display all the variables in a simple program, press the **Show Vars** button in the **Editor window**. This causes two windows to appear. The first window is called **Unit View: MAIN** which contains all the variables global to the main program. The second window, called **Stack Trace: Main Program**, contains the list of variables on the stack (the “local” variables). When the button is pressed, the check box in the **Show Vars** button is set. While the check box is set, whenever a program starts running, the two variable windows will be displayed.

Stop Displaying Variables

To stop displaying variables, close the **Unit View: MAIN** and the **Stack Trace: Main Program** windows. Pressing the **Show Vars** button will not hide the windows or stop them from updating. It is only used to force the two windows to appear.

When debugging a larger program, if you want execution to resume at full speed, you must close *all* the debug windows to stop OOT from updating them. As well, **Show Vars** must be turned off, otherwise the **Unit View: MAIN** and the **Stack Trace: Main Program** windows will be opened when execution begins again. You can locate all open debug windows by using the **Window** menu and checking the **Debug Windows** menu item to list all open debug windows.

Variables Window

The variable windows show the values of the variables. Individual records will have the word **record** by them where the value would appear. Following that, indented, will be the field names of the variable and their values.

Arrays have the word **array** instead of a value. Double clicking the line causes another window to appear that shows the value of the individual fields of the array. If it is a two-dimensional array, clicking on the horizontal slider at the bottom of the window switches between the different values for the first index.

The variable windows are updated every time a line is executed, so the user can see the values change as the program executes. As always, the user can pause the program and move step by step or start the program tracing.

The two variable windows are associated with each other, clicking on one window makes both windows visible.

Stack Window

The stack window also contains the calls to subprograms. The format is: the name of the function flush with the left side of the window, variables local to that subprogram indented. At the end of the subprogram's variables is a line of dashes. As the subprogram calls are made deeper and deeper, the list of variables grows downwards. As the program returns from a call, the contents of the stack window shrinks.

Note that each process has its own stack window.

Viewing Variables in Large Programs

Pressing the **Show Vars** button in a program with multiple units brings up the **Unit View: Main** window containing all the variables global to the main program. If the program uses multiple units, the **Unit List** window also appears. This window shows a list of all the non-predefined units used by the program. Double clicking any of the units in the list causes the **Unit View** window for that unit to appear. Thus the user can select specific units to see the variables of and leave all the other units unopened.

Likewise, if the program has multiple processes and the **Show Vars** button is pressed, the **Current Processes** window appears listing all the currently executing processes. Double clicking on any process in the list causes the **Stack Trace** window for that process to appear. Thus you can select which processes you wish to inspect and leave the other processes unopened.

Show Unit List

This command lists all the units created by the user (predefined units do not appear) used by the program. Double clicking on a unit in the unit list brings up the **Unit View** window for that particular unit.

Show Processes

This command shows all the processes and their status. Double clicking a process brings up the **Stack Trace** window for that particular process.

You can display all the variables in the entire program using the **Show Unit List** and **Show Processes** commands.

Tutorial

The following material is a very brief tutorial in creating, editing, running, printing, and saving a file. These are the initial activities required to start using OOT to write programs. As soon as you have mastered the basic activities, you are encouraged to read the other sections of this manual to find out the other commands at your disposal.

How to Create a New Window

If you have just started OOT for Windows, there may already be an **Editor window** labelled **unnamed**. You can start entering text in this window.

If you need to create another new window, select **New** from the **File** menu in the either the **Control Panel** or any other **Editor window**.

Note that when you save the program with a specified file name, the window title will reflect the new name.

How to Open an Existing File

In the **Directory Viewer**, locate the file that you want to open. If it is on a different drive, select the appropriate drive from the pop-up menu of drives found in the lower-right corner. To go into a directory (directories are listed first and enclosed in square brackets), double click on the directory name. To go up to the directory above, double click on the **[..]** in the list of files. To quickly move to the top level directory in the drive, double click on the **[\]**. Once you have found the file that you want to open, double click on it to open the file in an **Editor window**.

When you select **Save** from the **File** menu, the contents of the window will be saved into the file, regardless of the current directory.

How to Edit a Program

In the simplest case, you enter text by just typing. Press the **Enter** key to start the next line. You can erase text by pressing the **Backspace** key, which deletes the character directly behind the cursor (the small upward pointing arrow).

You can move the cursor by clicking the location in the **Editor window** that you wish to move the cursor to. At this point, you can insert text by typing, or delete text using the **Backspace** key.

Note that you can add blank lines at any point in the program by moving the cursor to the desired location and pressing the **Enter** key as often as required.

As the program expands beyond a single screen, use the scroll bars on the right-hand side to scroll the program up and down.

You can *select* a section of text using the mouse. Click down at the start of the text that you wish to select, and with the mouse button held down, drag the mouse to the end of the text that you wish selected and release the mouse button.

Once you have a *selection*, you can delete it by hitting the **Backspace** key or place a copy of it in the *paste buffer* by selecting **Copy** from the **Edit** menu. Once text is in the *paste buffer*, you can place a copy of the text at the location of the cursor by selecting **Paste** from the **File** menu.

When you have entered some portion of your program, you can *Paragraph* (indent) the program by pressing the **F2** function key or selecting **Paragraph file** from the **Edit** menu.

Note that this is just the beginning of the editor. Check the **Editor Menu** section and the **Editor Keystroke** commands for more commands to make editing programs easier.

How to Print a Program

To print a copy of your program, select **Print** from the **File** menu. Note that you must have already have a printer selected.

How to Save a Program

To save a copy of your program, select **Save** from the **File** menu. If you loaded the program from the disk, this saves the contents of the window in the file that you opened. If the window is *unnamed*, a dialog box labelled **Save As** appears. You can now enter the file name for this program.

If you opened this window from a file and wish to save the contents of the window in a different file, select **Save As** from the **File** menu. This opens the **Save As** dialog box which gives you the opportunity to specify a different file name. The original file will not be changed.

How to Run a Program

To run a program, press the **F1** function key or select **Run** from the **Run** menu. OOT then attempts to compile the program. Status messages appear at the bottom of the **Editor window** (and in the **Control Panel**) as the program is compiled.

If errors are found in the program then compilation is unsuccessful. The **Error window** appears with the list of errors in the program and the location in the program where the first error occurred is highlighted. Correct the errors in the program and then run the program again.

If the program finished compiling successfully (several seconds the first time, a second or two subsequent times), execution starts. A **Run window** appears in the lower-right corner and all output from the program appears there. You also enter all input to the

program here. Note that the **Run window** must be active in order for keyboard input to be sent to the OOT program.

You can halt execution of the program at any time by pressing the **Stop** button in the **Editor window**. Once the program has finished execution, the status message in the Editor window states “Execution finished”. At this point, you can print or close the **Run window**. You can close the **Run window** by double clicking on the **Control** menu (the box in the upper-left corner of the window with the horizontal bar running through it).

How to Print a Program’s Output

To print the output of the **Run window**, make certain that the **Run window** is selected and the press Control-P (that is the “Control” key and the “P” key pressed simultaneously). If you need to print out more than one screen’s worth of output, see the section labelled **Input Redirection**.

How to Quit OOT

To finish using OOT, select **Quit OOT** from the **File** menu of either any **Editor window** or the **Control Panel**.

Chapter 3

OOT for Macintosh

Installing OOT for Macintosh

Minimum System Requirements

Object Oriented Turing for Macintosh requires:

System Software:	System 7.0 or greater
Hardware:	68020/68030/68040/PowerPC Macintosh 3MB of free RAM 5 Mb of disk space

If you are uncertain as to which version of the system software you are running, switch to the Finder and select **About This Macintosh** under the **Apple** menu. In the upper right of the dialog box that appears, the version of the system software you are currently running is displayed. This must be version 7.0 or greater.

The only Macintoshes that don't have a processor capable of running OOT for Macintosh (MacOOT) are the Macintosh 128, 512, Plus, SE, Classic, original Portable and PowerBook 100.

If you are uncertain as to the amount of memory on your system and whether you need to activate virtual memory, switch to the Finder and select **About This Macintosh** under the **Apple** menu. The figure titled **Largest Unused Block** is the amount of free memory available. If the line indicates that less than **3,000 KB** of memory are free, you will need to enable virtual memory (detailed below). Note that quitting applications also frees up memory.

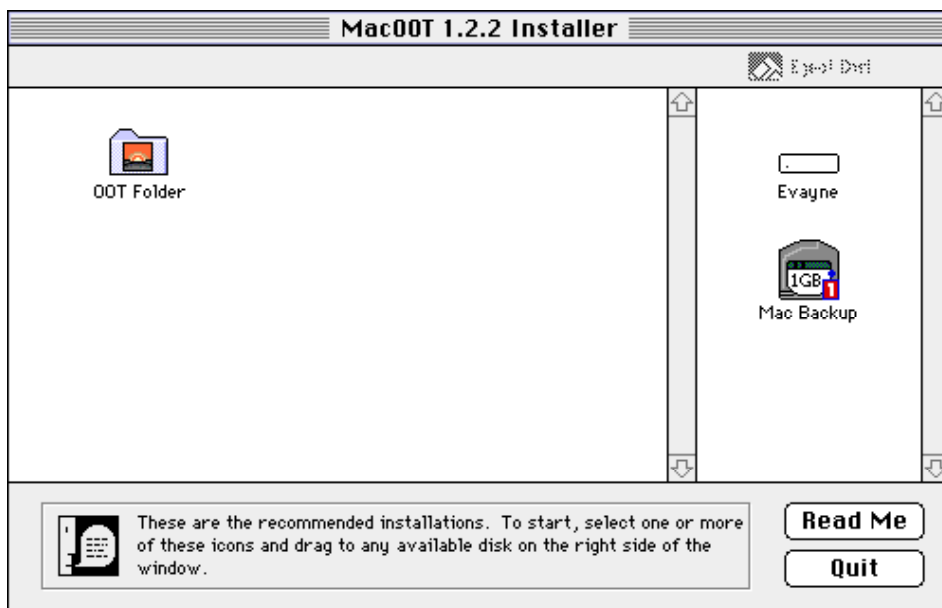
Installing Under Macintosh

- 1) Place the original **Object Oriented Turing for Macintosh** disk into the disk drive.

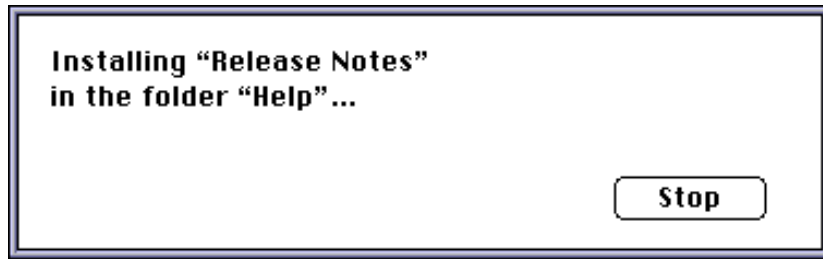
- 2) Double click on the icon labelled **MacOOT 1.2.2 Installer**. Note that the version number will probably be different.



- 3) A window shows up showing the OOT Folder on the left hand side and all the disks onto which OOT can be installed on the right.



- 4) Drag the OOT folder onto the disk where you want OOT for Macintosh to be installed.
- 5) An information box comes up telling you which file is being installed.



- 6) When all the files are installed, a dialog box comes up allowing you to do another installation or continue. Select the **Quit** button (or press **Return**).



- 7) Another dialog box appears telling you which version of MacOOT is being installed.



- 8) If this is the first time you have installed the software, a reminder to send in your registration card appears. **Please send in your registration card!**
- 9) Another dialog box appears. This one has spaces to enter your name, address and registration number. (Note the Academic version has a space only for the institution and registration number.) Please fill it in. If this is not the first time the software has been installed, all fields but the registration number will already be filled in and can't be changed. The user must then only fill in the registration number.

Important: For the commercial version, your registration number is on a sticker labelled OReg# followed by 8 digits. The 8 digits are your registration number. The sticker will be in one of two places:

- 1) In the lower left hand corner on the first page of the *Turing Reference Manual*.
- 2) On the actual disk itself (if purchased without the *Turing Reference Manual*).

For the academic version, the registration number is found on the loose leaf release notes that accompanied the diskette. The academic registration number is 8 characters, an 'A' followed by 7 digits.



The image shows a dialog box titled "OOT Registration Information". It contains five text input fields and two buttons. The fields are labeled "User Name", "Organization", "Street Address", "City/Province", and "Registration Number". The "User Name" field contains "Tom West", "Street Address" contains "35 Starwalk Crescent", "City/Province" contains "Toronto, Ontario, Canada", and "Registration Number" contains "17748345". The "Organization" field is empty. At the bottom right are "Cancel" and "OK" buttons.

OOT Registration Information	
User Name	Tom West
Organization	
Street Address	35 Starwalk Crescent
City/Province	Toronto, Ontario, Canada
Registration Number	17748345
<div>Cancel OK</div>	

- 10) Once you have filled in all the appropriate fields and clicked **OK**, the dialog box will disappear and registration is now complete. You can now run OOT for Macintosh by double clicking on the **OOT for Macintosh** icon found in the **OOT Folder** on your hard disk. Please keep the original diskette safe in case you need to reinstall the software.

De-installing OOT for Macintosh

- 1) Drag the **OOT Folder** from your hard disk into the trash.
- 2) Open up the **Preferences** folder found in the **System Folder** and drag the **MacOOT Preferences** file found there into the trash.

Using OOT for Macintosh

Starting MacOOT

Once OOT is installed, you can launch the OOT for Macintosh application.

On the hard drive where you installed OOT for Macintosh (MacOOT), you will see a folder called **OOT Folder**.



Open up the folder by double clicking on it. The folder contains a number of files and folders and an icon labelled **MacOOT**.

To start MacOOT, double click on the icon labelled **MacOOT**. You can also launch MacOOT by double clicking on any MacOOT file. MacOOT files look like this.

MacOOT Environment

Once MacOOT is launched, you are presented with the initial alert dialog box which displays the version of OOT and the name of the person for whom the software is licensed. MacOOT also compiles the built in modules at this time. The alert box disappears after a few seconds.

If MacOOT is double clicked to start it up, it displays an untitled **Editor window** in the upper-left corner, otherwise it opens up the MacOOT file that the user clicked on into an **Editor Window**. Along with **Editor windows**, which contain user programs, there are several other kinds of windows in MacOOT: **Run windows** which are the windows

produced when your program runs, an **Error Window** which contains a list of all the errors found in your program, a **Status window** which contains the messages that tell you what is being compiled or executed, the **Debugging Controller window**, which is used to step, trace and view variables in a program, and **Debug windows** which are windows containing debugging information.

What follows is a description of the windows that appear in MacOOT.

MacOOT Windows - Editor Window

An **Editor window** is used to enter and edit programs or to display text files.

The title bar of an **Editor window** contains the name of the file displayed in the window. If the file has not yet been named, it will appear as *Untitled*.

Editor Window

The **Editor window** is where all programs are displayed and modified. If the program extends to the bottom of the window, the right hand scroll bar is activated and you can use it to scroll through the program. If any line in the program is longer than the width of the window, then the bottom scroll bar is activated. You can change the default size of an **Editor window** in the Preferences dialog box.

MacOOT Windows - Run Window

Run Window Displaying Output from %oot/examples/widget/widget.dem

All output from an OOT program appears in the **Run window**. In the example above, there is a large amount of graphical output appearing in the window. (Note that what appears as a menu bar at the top of the window is produced by the output of the OOT program being executed. It is *not* part of the **Run window** itself.)

By default **Run windows** are 25 columns by 80 rows, although you can change this using the OOT Preferences. All output appears in the **Run window** except output being sent to files. It is not yet possible to have multiple **Run windows** open with output being sent to each window, although this will be implemented in a future version of MacOOT. At that point, a call to **Window.Open** will open another **Run window**.

To print the contents of a **Run window**, select the **Print** menu item from the **File** menu or press **⌘P** when the window is active (in front).

To save the contents of a **Run window**, select **Save As** from the **File** menu when the Run window is active. If the **Run window** is in graphics mode, the contents are saved as a PICT format graphics file. If the **Run window** was in text mode, then the contents of the window and any text that scrolled off the top of the window are saved in a text file. You

can also save the output of a run by redirecting your output to a file. This is discussed in detail in *Running Programs - Input and Output Redirection*.

MacOOT Windows - Debugger Window

Debugger Window - Unit List

Debugger Window Showing Variable Contents

Debugger Window Showing Array Contents

Whenever the user turns on **Show Vars** using the **Debugger Controller**, OOT displays the output in **Debugger windows**. There are three types of debugger windows. There is the **Unit List Debugger window**, which displays all the units found in the currently executing program. There are **Variable Display Debugger windows**, such as the second debugger window displayed. Clicking on objects or arrays in these windows will cause another Debugger window to open displaying the contents of the object or array. Lastly, there are **Array debugger windows**, such as the third window displayed. These display the contents of arrays. For more information about viewing variables see *Showing Variables*.

Records are displayed with the fields of the record indented in order to show their place in the record. Like a regular variable, a field of a record containing an object or an array can be double clicked on to display the object or array in a separate window.

MacOOT does not currently have the ability to display two dimensional arrays. In order to view variables in such an array, the array must be declared as an “array of arrays” (e.g. **array 1 .. 10 of array 1 .. 5 of int**).

MacOOT Windows - Debugging Controller

The **Debugging Controller** is a unique window in MacOOT (i.e. you can only have one of **Debugging Controller** visible at a time). The Debugging Controller is also a floating window, which means that it always appears on top of all other windows, even when it is not active.

The **Run** button compiles and executes the *main program* (either the active window or a specified window selected by the **Fix Main Program** menu item in the **File** menu). When a program is executing, this button changes to **Stop** and stops program execution when pressed. The next button is the **Pause** button. This button pauses a program that is executing. When a program is paused, you can see the execution line, view variables or set breakpoints (see *Stepping and Tracing* and *Showing Variables* for more information). When a program is paused, the **Pause** button changes to **Resume**. Pressing this button causes execution to continue from where it left off.

The **Step** button causes OOT to execute the next statement of a paused program (usually highlighted) and then pause, highlighting the line next line to be executed. If the statement to be executed is a call, an importation of a module, or is creating an object, OOT will step into the subprogram (or module or class initialization). This button is inactive when the program is executing and is not paused.

The **Step Over** button acts like **Step** except that if the current line is a call such as to a subprogram, pressing **Step Over** will cause the call to be made and execution to be paused at the line following the call.

The **Step Return** button operates like **Step** or **Step Over**, except that execution continues until the subprogram, module, or class initialization completes. The line highlighted will be the subprogram call. Pressing **Step** advances execution to the following line.

The **Tracing** check box will toggle tracing on and off, as noted by the check box in the button. If tracing is on, whenever a program is executing the next line to be executed will be highlighted.

The speed of execution is controlled by the **Trace Speed slider** situated to the right of the **Tracing** check box. This slider controls how fast the program executes while tracing is on. The more to the left the slider is placed, the longer OOT will pause on each line before executing the next line. When slid to the far right, OOT will execute the program (while displaying the next line to be executed) without pausing. See *Stepping and Tracing* for more details.

The **Show Vars** check box is similar to the **Tracing** check box in that it toggles showing variables on and off. When selected, this check box causes the currently executing program to display the list of units (if more than one), the global variables of the main unit, and the

stack of the primary process. See *Showing Variables* for more information. Note that even when the **Show Vars** check box is pressed again in order to turn this feature off, any still visible **Debugger windows** will be updated, slowing execution down. In order to stop the debugger from continuing to display variables, you must have turned **Show Vars** off and closed all open **Debugger windows**. (You can close all the Debugger windows and hide the **Debugging Controller**) by selecting the **Hide Debugger** menu item from the **Debug** menu.

Beneath the buttons is the status message. This displays the current execution status allowing the user to see at a glance whether the program is executing, paused or has been halted.

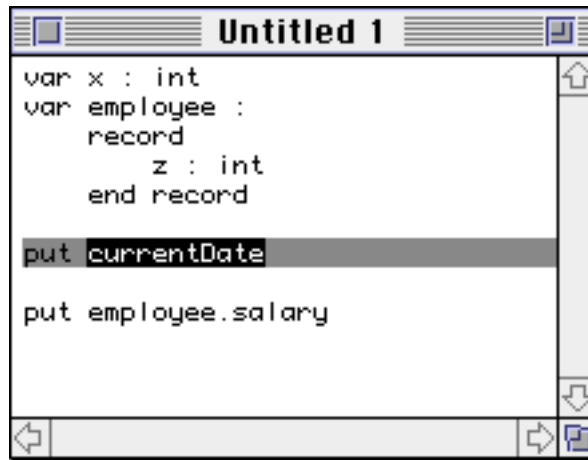
MacOOT Windows - Error Viewer

This window lists any error messages that occur while compiling or executing the program.

Error Viewer

The body of the window displays all the error messages concerning the program. Each line contains the text of the error followed by the line number and filename that the error occurred on. Normally the line and file information is not visible as it is off the right hand side of the window. You can scroll the Error Messages window to the right to get the line and file information.

To go to the location of an error in the program, double click on the line in the **Error Viewer** containing the error message. The line will be highlighted and the file containing the error will be made active and the line containing the error will be highlighted.



Editor window with error highlighted


Editing Text - Keyboard Editing Commands

There are a variety of keyboard commands that can be used in an **Editor window**. By learning these commands, you can edit your file more quickly and efficiently.

Moving the cursor:

Up Arrow Key	Move cursor up a line.
Down Arrow Key	Move cursor down a line.
Left Arrow Key	Move cursor left a character.
Right Arrow Key	Move cursor right a character.
Option+Left Arrow	Move the cursor left a word.
Option+Right Arrow	Move the cursor right a word.

Editing the text

Delete	Delete character to left of cursor.
F1	Run the <i>main program</i> .
F2, 	Paragraph program.
F8	Get one line reference on selected keyword.
F9	Get <i>Turing Reference Manual</i> entry on selected keyword.

Editing Text - Selection, Cutting and Pasting

To move or duplicate large sections of text, *select* text, move it to the *paste buffer* (also called the *clipboard*) and then *paste* it into the text. Text can also be selected in order to perform other operations, such as commenting out or uncommenting a section of program, deleting a section, and so on.

Selection can be accomplished using the mouse or the keyboard. To select text using the mouse, click at the text position where the selection is to start. Holding the mouse button down, drag the mouse to the location in the text where the selection is to end. The selection grows from the starting position (called the *anchor point*) to the current mouse position. You can see the highlighted text easily as the selection is displayed in the system highlight color (specified in the **General** control panel).

To select text using the keyboard, use the cursor keys to move the text cursor to where the selection is to begin. This location becomes the selection's anchor point. Holding the shift key down, use the arrow keys to move the text cursor around. The text between the anchor point and the current text cursor position is highlighted as the selection.

Once the text is selected, you can perform various operations on the selection. The simple operations are **Cut**, **Copy**, and **Paste**. **Cut** copies the selected text to the paste buffer and then deletes the text in the document. Use this when you want to move text to a different location in the document. **Copy** copies the selected text to the paste buffer but does not affect the location. **Paste** places the contents of the paste buffer at the text cursor location.

Note that using **Paste** does not delete the contents of the paste buffer. It is possible to paste the contents of the paste buffer several times without cutting or copying each time.

Editing Text - Searching and Replacing Text

When editing documents, it is often helpful to be able to either find or replace occurrences of one string for another in a file. This capability is called *search and replace*. To find a particular line in the text, select **Find** from the **Search** menu. The **Find** dialog box appears giving you the opportunity to specify the text to search for.

Find Dialog Box

Enter the text to be searched for in the text box labelled **Find What?**. The text in the text box is called the *search text*.

Beneath the *search* text are two radio buttons labelled **Literal** and **Pattern Match**. You can either search for a literal match (find the text in the **Editor window** that exactly matches the *search text*) or using pattern match, in which case you can enter any regular expression in the text box and OOT will match anything matching the regular expression.

There are two check boxes to the right of the radio buttons. The **Search Backwards** text box causes OOT to look for the *search text* backwards starting from the beginning of the current selection, rather than forwards from the end of the current selection.

When the **Wrap-Around** check box is checked, whenever a forward search reaches the bottom of a file, the search is continued from the top (although it stops if it searches the entire file and does not find anything). Likewise, if **Wrap-Around** is selected, any backwards search that hits the top of the file continues searching from the very bottom.

Pressing the **Find** button (or pressing Return) causes the Find to take place. If the *search* text is found, then the **Editor window** is scrolled so that the matching text is visible and the matching text itself is highlighted. Once you have found the text, you can search for the next occurrence of the *search text* by typing ☐G or selecting **Find Next** from the **Search** menu. No dialog box is displayed for **Find Next**.

To search for another occurrence of text that appears in the **Editor window**, highlight the text in the **Editor window** to search for using the mouse and then press ☐H or select the **Find Selection** menu item from the **Search** menu. If you press the shift key with either ☐G or ☐H, the direction of the search is reversed. (Handy for "backing up" when you are searching for a particular occurrence of the text and you skip over the desired text by accident.)

To replace text, you use the **Change** command (select **Change** from the **Search** menu or press ☐L. This command brings up the Change dialog box. The Change dialog box has space to enter the text to search for (the *search text*) and the text to replace the *search text* with (the *replace text*).

Change Dialog Box

The radio buttons and check boxes are identical to the **Find** dialog box. The buttons at the bottom are different. For one, the **Change** dialog box doesn't disappear when a button is pressed. When the user selects **Find Next**, OOT searches for the next occurrence of the *search text*. When it finds it, it highlights the matching text in the **Editor window** and enables the **Change, then Find** and **Change** buttons, making **Change, then Find** the default button.

The user can then either change the matching text to the *replace text* and immediately search for the next occurrence of the *search text* (by selecting **Change, then Find**), change the matching text to the *replace text* and observe the results (by selecting **Change**), change every occurrence of the *search text* in the **Editor window** to the *replace text* (by selecting

Change All) or skipping the current match and finding the next match (by selecting **Find Next**).

If there is no text specified for the *replace text*, then a doing a change is the same as deleting the matching text.

You can also use **⌘F** for **Find Next**. To hide the **Change dialog box**, just click the close button on the window.

Running Programs -

Input and Output Redirection

To run a program, select the **Run** menu item from the Run menu or press **⌘R**. You can also press **F1** when an **Editor window** is selected. The Run command compiles and executes the selected window or a fixed selected program. See *Running Programs - Programs with Multiple Parts* for more details on how to select the file to be run.

Once the program is running, a **Run window** will be created and the output of the program displayed in the **Run window**. You can print and save the output in the **Run window** using the **Print** and **Save As** commands.

You can also specify that input should come from a file instead of the keyboard. Both these operations are performed by selecting **Run with Arguments** from the **Run** menu. When you do so, the **Run with Arguments** dialog box appears.

Arguments and Redirection Dialog Box

This dialog box has several parts. The first part is the **Arguments** field. You can enter any text in the **Arguments** field and the text will be passed as command line arguments to the program when it executes. The program can read the command line arguments using the OOT predefined subprograms **nargs** and **fetcharg**.

In the **Input From** section you choose one of the three radio buttons. If you select either **File** or **File with Echo**, then an **Open File** dialog box comes up asking you to select the file that you want to use as input (i.e. with **get** statements) to the program. If you choose **File**, any input read from the file will *not* be echoed to the screen. If you choose **File with Echo**, then anytime a line of input is read, it is also echoed to the output window.

In the **Output From** section, you choose one of three radio buttons. If you select either **File** or **Screen and File**, then a **Save File** dialog box comes up allowing you to specify the name of the file the output should be sent to. Note that only text output (i.e. **put** statements) will be redirected to a file. No graphics will be redirected. If the user selected **File**, then the output will be sent to the file and no text output will be visible by the user of the program.

If the user selected **Screen and File** then the text output will appear on the screen as well as being sent to a file.

In future versions of the software, there may be a **Compile** menu item in the **Run** menu. This will be used for producing standalone versions of compiled programs (essentially applications that you can run without having OOT present). When this becomes available, an item in will appear in the **Help** menu explaining the use of this feature.

Running Programs - Programs with Multiple Parts

In OOT, editing and executing a program consisting of a single file (no **import** or **include** statements) is straightforward. When the **Run** or **Run with Arguments** command is given, OOT compiles and executes the contents of the active **Editor window**, or if no **Editor window** is active, the last active **Editor window**. The program that is compiled and executed when the run command is given is called the *main program*.

In cases where a user is editing and executing programs consisting of several units (usually a main program and several modules or classes in separate files), it is not always obvious which file should be run. Running the currently active **Editor window** will not work if the active window is an **Editor window** containing one of the classes the program uses instead of the main program itself.

Rather than always having to activate the window containing the main program before selecting **Run**, OOT provides an alternative method of specifying which program to compile when the run command is selected.

This is done by setting the *main program* to be either *fixed* or *floating*. When the *main program* is *floating* (the default method), the *main program* is either the currently active **Editor window**, or if no **Editor window** is active, the last active **Editor window**. If the *main program* is *fixed* then the *main program* is a specific file that the user previously chose.

For example, a file called *database.t* uses two units *readdata.tu* and *writedata.tu*. The user would specify *database.t* as the *main program*. Then regardless of which file is being edited, when the run command is given, OOT compiles and executes *database.t*.

If the user later wants to run a different program, she or he sets the *main program* to be *floating*. OOT then compiles and executes the active **Editor window**.

The following commands relate to selecting the *main program*. The **Run** command compiles and runs the *main program*. The name of the current *main program* can be found in the **Run** menu where the **Run** menu item is either labelled **Run** if the main program is *floating* or **Run "filename"** if the *main program* has been *fixed* to be *filename*.

The **Run** and **Run with Arguments** commands compile and execute the *main program*.

The **Fix Main Program** menu item in the **File** menu fixes the main program to be the contents of active **Editor window**. The **Float Main Program** causes the *main program* to *float* once again.

Menu Commands

Apple (⌘)

About MacOOT: Display the current version and creator of OOT and MacOOT.

File

New: Open an empty untitled **Editor window**.

Open: Bring up the **Open** dialog box to allow the user to open a file into an **Editor window**.

Close: Close this **Editor window**. If the window has been changed since it was last saved, there will be an opportunity to save it.

Save: Save the contents of the **Editor window** in the file associated with the window. If the window is unnamed, bring up a **Save As** dialog box to allow the user to select the name and directory of the file to be saved.

Save As: Bring up the **Save As** dialog box. Save the contents of the **Editor window** in the file selected. Note that this changes the file associated with the window to be the new file.

Revert to Saved: Discard any changes since the program was last saved.

Page Setup: Configure the printer.

Print: Print the program on the printer selected by Chooser.

Fix Main Program: *Fix* the active **Editor window** to be the *main program*. Whenever a the user selects **Run**, either from menu or keyboard shortcut, this will be the program that is run until another file is made the main program or **Float main program** is selected. See *Running Programs - Programs with Multiple Parts*.

Float Main Program: Cause the main program to float again. When the main program is floating, the active or last active **Editor window** will be the one that is run when the run command is given. See *Running Programs - Programs with Multiple Parts*.

Preferences: Display a series of preferences. There are three debugging preference dialogs which can be toggled between using the Next and Previous buttons. These are the Editor, Printing and Running preferences. Using the **Set Default** button sets all the preferences for all three dialog boxes to their default settings. Pressing **Cancel** means that the preferences are not changed. Pressing **OK** sets the preferences to whatever the user selected.

Memory Usage: Displays a dialog box containing the amount of free memory available under MacOOT. Use this to check if you are running out of memory (in which case, increase the memory partition of MacOOT using the Finder).

Quit : Quit the OOT environment. Close all OOT windows. Ask the user whether she or he wishes to save the contents of **Editor windows** that have been modified.

Edit

Undo: Undo the last operation, restoring the contents of the window to what they were before the operation.

Cut: Copy the selection to the paste buffer and then delete the selection. See *Selection, Cutting and Pasting* for more details.

Copy: Copy the selection to the paste buffer. See *Selection, Cutting and Pasting* for more details.

Paste: Paste the contents of the paste buffer at the text cursor. If there is a selection, the paste buffer replaces the selection. See *Selection, Cutting and Pasting* for more details.

Clear: Delete the selected text. Don't place it in the paste buffer.

Select All: Selects all the text in the active **Editor window**.

Comment: "Comment out" all lines in the selection. It places "%" at the beginning of all lines in the selection. This is useful for temporarily disabling sections of the program while testing it.

Uncomment: "Uncomment" all lines in the selection. Removes the "%" at the beginning of any lines in the selection. This command is used in conjunction with the **Comment range** command to reenables sections of the program that were disabled using the **Comment range** command.

Structure completion: This feature is not yet implemented. Future versions may implement it, so it is documented here. If the text cursor is at an OOT program construct (for example **loop**, **case**, and so on.), this command places the end-construct after the construct, adds a blank line between the two, and moves the text cursor there, indenting the line by four spaces. For example, if you have:

```
loop^
```

where ^ represents the text cursor, selecting **Structure completion** produces the following:

```
loop
  ^
end loop
```

Structure match: This feature is not yet implemented. Future versions may implement it, so it is documented here. If the text cursor is at an OOT program construct (for example **loop**, **case**, etc.), or at an end-construct (for example **end loop**, **end case**, etc.), this command moves the text cursor to the end of that particular construct. If at the end of the construct, **Structure match** moves to the beginning of the construct. For example, if you have:

```

for i : 1..20^
  loop
    loop
      put "A"
    end loop
  end loop
end for

```

where ^ represents the text cursor, selecting **Structure match** moves the cursor to the beginning of the **end for**:

```

for i : 1..20
  loop
    loop
      put "A"
    end loop
  end loop
^end for

```

selecting **Structure match** a second time moves the cursor to the beginning of the first line.

Paragraph: Indent the file, splitting lines that are too long. This command formats the file, indenting constructs properly and splitting lines that are longer than 78 characters or have more than one statement on them (if possible). For example, if you have:

```

for i : 1..20
loop
if x = 3 then put "A" end if
end loop
end for

```

selecting **Paragraph** produces the following:

```

for i : 1..20
  loop
    if x = 3 then
      put "A"
    end if
  end loop
end for

```

Search

Find: Find the next occurrence of a specified string. Brings up the **Find** dialog box. See *Searching and Replacing Text* for more information.

Find Next: Find the next occurrence of a previously specified string. Note that Shift+⌘G does a search in the opposite direction from previously specified in the Find Box. (This is a handy way of going back if you press ⌘G once to often.)

Find Selection : This will find the next occurrence of the currently selected text. Dimmed if there is no selection or if the selection is too long. Shift+⌘H will search backwards for the same string.

Find Next Error: Highlight location of the next error in an **Editor window**.

Change: Replace specified text with another specified text. You can also change all occurrences.

Go To Line: Jumps to a specified line in the window. Useful if you have a listing with line numbers on it.

Mark

This window contains a list of all the procedure, functions, modules, monitors, processes and classes of the current **Editor window**. It is updated every time the window is opened or paragraphed. Selecting on an item in the window causes the declaration of the item to be displayed. (i.e. the **Editor window** scrolls to the beginning of the item selected.)

Windows

This menu contains the list of all open windows in OOT. Non-unique windows are found in submenus labelled **Editor Window**, **Run Window**, **Help Window** and **Debug Window**. Selecting a menu item from this menu causes the corresponding window to come to the front and to become active.

Tile Windows: Resize all the **Editor windows** so that they cover the entire screen.

Stack Windows: This stacks the windows on top of each other, offsetting each one down and to the right of the one beneath it.

Editor Window: A sub-menu containing all the open **Editor windows**.

Run Window: A sub-menu containing all the open **Run windows**.

Help Window: A sub-menu containing all the open **Help windows**.

Debug Window: A sub-menu containing all the open **Debug windows**.

Change Dialog: Bring the change dialog box to the front. If the dialog box isn't visible, make it visible.

Debugger Controller: If the debugger controller isn't visible, make it visible.

Error Window: Brings the error window to the front. If the error window isn't visible, make it visible.

Status Window: Brings the status window to the front. If the status window isn't visible, make it visible.

Debug

Show/Hide Debugger: This either makes the **Debugger Controller** visible or hides the **Debugger Controller** and all the **Debugger windows**.

Show/Hide Breakpoints: This will cause all program windows and the unit window to have a column on the left hand side displayed (or hidden). Clicking by a line in the left hand column causes a breakpoint to be displayed (a red diamond). Clicking again causes the breakpoint to be removed. You can place or remove breakpoints while the program is executing. See *Show Setting Breakpoints*.

Clear Breakpoints: This erases all breakpoints.

Show unit list: Display the list of all units (main unit, modules, and classes) used by the currently executing program. See *Show Variables*.

Show processes: Display the list of all processes used by the currently executing program. See *Show Variables*.

Show queues: Display the list of all processes used by the currently executing program by the queue that they are in (the queues can be *executing*, *waiting for event*, *paused*, and so on.).

Show allocated objects: Show some objects allocated by the currently executed OOT program. These objects will include file stream numbers, directory stream numbers, Pics, and other elements from the predefined modules in later versions of the OOT software.

Show Memory Usage: This is not currently implemented. When implemented, it will bring up a debugger window displaying all allocated memory (anything using the **new** keyword) including the location in the source.

Show Execution Line: This command highlights the line in the source that the selected process is currently executing. Useful for seeing what section of code is being executed without pausing the program.

Show Definition: This command highlights the line in the source that the selected item was defined on.

Run

Run: Run the main program. If the main program is floating, run the active **Editor window**. You can always determine the current main program by the fact that the name is in quotes in the **Run** menu (as in **Run "filename"**).

Run with Args: Run a program (as above) but bring up the **Run with Arguments** dialog box. See *Running Programs - Input and Output Redirection*.

Stop execution: Terminate execution of compilation of the program immediately.

Set preprocessor symbols: This feature is not yet implemented. It is documented for later use. Bring up a dialog box allowing the user to specify all the preprocessor symbols for use with the **#if** statements in an OOT program.

Compiler reset: Reset the compiler. Throw away all compiled units.

Help

Help with Editor: Display a text file containing help on the editor. In later versions of OOT, if there are commands in the editor that are not documented here, check the Editor help file using this command for instructions on use.

Help with Debugger: Display a text file containing help for the debugger.

Keyword usage: If the text cursor is at an OOT predefined subprogram or part of the OOT language, display a one line summary of the *Turing Reference Manual* entry in an alert box. (Shortcut **F8**.)

Keyword reference: If the text cursor is at an OOT predefined subprogram or part of the OOT language, this display a window containing the *Turing Reference Manual* entry on the item. (Shortcut **F9**.)

OOT Reference: A sub-menu containing help files about the release notes, the OOT language in comparison to Turing Plus and converting Turing Plus programs to OOT.

Information: This feature is not yet implemented.

Debugging - Stepping and Tracing

It is very useful to be able to follow the execution of a program visually. OOT lets you see each line of a program as it is executed. By doing so, you can see exactly how the program *is* executing, compare it with how it *should be* executing, and discover bugs.

As well, viewing the execution of the program can be an instructional aid to learning how various language constructs operate.

Stepping and tracing provide two ways of seeing a program execute. Stepping is when OOT executes a small chunk (usually one line) of a program and then waits for another command. Tracing is when a line being executed is highlighted for a set amount of time, then the next line, and so forth. In Tracing, execution continues until the user gives a command to stop execution (or the program terminates). In stepping, the user must give a command to continue after each line (or chunk) of code.

In MacOOT, to do any stepping or tracing, you need to bring up the **Debugger Controller**. This is done by selecting **Show Debugger** from the **Debug** menu.

Tracing

To start OOT tracing a program, press the **Trace** button in an **Debugger Controller**. If a program is already executing, OOT starts highlighting the line being currently executed and continues to execute the program. Otherwise, OOT starts tracing execution the next time a program is run. To turn off tracing, press the **Trace** button again. (Note that the check box becomes empty.)

You can control the speed at which tracing occurs by adjusting the slider labelled **Trace Speed** on the **Debugger Controller**. To have each line executed more quickly, move the slider to the right. To slow execution down, move it to the left. Anytime you want execution

to resume normally, click the **Trace** button again. You can go back and forth turning tracing on and off as you deem appropriate.

Stepping

You can also examine execution one line at a time (i.e. stepping). While the program is running, click on the **Pause** button in the **Debugger Controller** (note that it changes to read **Resume**). Once the program is paused, the three buttons in the **Debugger Controller** labelled **Step**, **Step Into**, and **Step Return** are enabled. Clicking the **Step** button causes OOT to execute one line of code (the next line will be highlighted). If you are at a call to a subprogram, the **Step** button enters the subprogram and highlights the first line of it. If you wish to trace execution over the subprogram (in other words, execute the subprogram all at once and pause at the statement after the call), click on the **Step Over** button. As well, if you are in a subprogram and wish to quickly skip the rest of a subprogram, click the **Step Return** button, which executes the rest of the procedure and pauses execution at the statement following the call to the subprogram.

Note that pressing **Step Return** in the main program is the same as pressing the **Step** button.

Note that if you are stepping over multiple lines (by doing a **Step Over** or a **Step Return**) while tracing is turned on, OOT will not highlight the individual lines being executed.

When stepping through a program, you can always continue full speed execution by making certain that tracing is turned off and pressing the **Resume** button.

Breakpoints

MacOOT has the ability to dynamically set breakpoints (points where execution of the program stops and allows the user to start stepping or tracing). To set a break point, the user selects **Show Breakpoints** in the **Debug** menu. All **Editor Windows** and the **Unit List Debugger Window** display a column on the left hand side. Clicking in that column to the left of a line causes a red diamond to be displayed adjacent to the line. When execution reaches that line, then the program pauses just as if the use has pressed the **Pause** button.

To remove a breakpoint, click on the red diamond and the breakpoint is removed. Setting a breakpoint on a unit (from within the **Unit List Debugger Window**) will cause execution to pause whenever any line of that unit is executed.

To clear all the breakpoints from a program, select **Clear Breakpoints** from the **Debug** menu.

Editor window with two breakpoints set

The user can set a static breakpoint in the program. This is done by placing the **break** statement anywhere in the program. When the program reaches the **break** statement, execution is paused just as if the user pressed the **Pause** button at that point. The user can then issue any of the stepping, tracing or view variable commands, or press the **Resume** button to continue execution. The user can have as many **break** statements in the program as he or she wishes. Execution will pause at each **break** statement.

Showing Variables

OOT is capable of showing all the variables in a program. To understand how OOT displays the variables, you must understand the way the variables are grouped. First there are unit variables, grouped one window per unit, and then there are stack variables, grouped one window per process. A program consisting of one unit (no **import** statements) and one process (no concurrency) would have one window of unit variables and one window of stack variables. When observed in OOT, the two debugging windows would be considered the "global" variables and the "local" variables.

A unit is either the main program or it is a module or class in a separate file with the keyword **unit** at the top of the file. OOT programs use units using the **import** statement. Unit variables are those in a given unit that are declared outside of any subprograms. These variables are common to all subprograms in a unit. In the main program unit, they are known as global variables.

A process is one thread of execution. Each process has its own local variables. The stack variables for a particular process consist of all the local variables in the subprograms that the process is currently in. In simple terms, stack variables are called local variables.

Displaying Variables

To display all the variables in a simple program, press the **Show Vars** button in the **Debugger Controller**. This causes two windows to appear. The first window is called **Unit View: MAIN** which contains all the variables global to the main program. The second window, called **Stack Trace: Main Program**, contains the list of variables on the stack (the "local" variables). When the button is pressed, the check box in the **Show Vars** button is set. While the check box is set, whenever a program starts running, the two variable windows will be displayed.

Stop Displaying Variables

To stop displaying variables, close the **Unit View: MAIN** and the **Stack Trace: Main Program** windows. Pressing the **Show Vars** button will not hide the windows or stop them from updating. It is only used to force the two windows to appear.

When debugging a larger program, if you want execution to resume at full speed, you must close *all* the debug windows to stop OOT from updating them (select **Hide Debugger** from the **Debug** menu). As well, **Show Vars** must be turned off, otherwise the **Unit View: MAIN** and the **Stack Trace: Main Program** windows will be opened when execution begins again.

Variables Window

The variable windows show the values of the variables. Individual records will have the word **record** by them where the value would appear. Following that, indented, will be the field names of the variable and their values.

Arrays have the word **array** instead of a value. Double clicking the line causes another window to appear that shows the value of the individual fields of the array. If it is a two-dimensional array, clicking on the horizontal slider at the bottom of the window switches between the different values for the first index.

The variable windows are updated every time a line is executed, so the user can see the values change as the program executes. As always, the user can pause the program and move step by step or start the program tracing.

The two variable windows are associated with each other, clicking on one window makes both windows visible.

Stack Window

The stack window also contains the calls to subprograms. The format is: the name of the function flush with the left side of the window, variables local to that subprogram indented. At the end of the subprogram's variables is a line of dashes. As the subprogram calls are made deeper and deeper, the list of variables grows downwards. As the program returns from a call, the contents of the stack window shrinks.

Note that each process has its own stack window.

Viewing Variables in Large Programs

Pressing the **Show Vars** button in a program with multiple units brings up the **Unit View: Main** window containing all the variables global to the main program. If the program uses multiple units, the **Unit List** window also appears. This window shows a list of all the non-predefined units used by the program. Double clicking any of the units in the list causes the **Unit View** window for that unit to appear. Thus the user can select specific units to see the variables of and leave all the other units unopened.

Likewise, if the program has multiple processes and the **Show Vars** button is pressed, the **Current Processes** window appears listing all the currently executing processes. Double clicking on any process in the list causes the **Stack Trace** window for that process to appear. Thus you can select which processes you wish to inspect and leave the other processes unopened.

Show Unit List

This command lists all the units created by the user (predefined units do not appear) used by the program. Double clicking on a unit in the unit list brings up the **Unit View** window for that particular unit.

Show Processes

This command shows all the processes and their status. Double clicking a process brings up the **Stack Trace** window for that particular process.

You can display all the variables in the entire program using the **Show Unit List** and **Show Processes** commands.

Tutorial

The following material is a very brief tutorial in creating, editing, running, printing, and saving a file. These are the initial activities required to start using OOT to write programs. As soon as you have mastered the basic activities, you are encouraged to read the other sections of this manual to find out the other commands at your disposal.

How to Create a New Window

If you have just started OOT for Macintosh, there may already be an **Editor window** labelled **untitled**. You can start entering text in this window.

If you need to create another new window, select **New** from the **File** menu.

Note that when you save the program with a specified file name, the window title will reflect the new name.

How to Open an Existing File

Select **Open** from the **File** menu. This brings up the standard Macintosh file selector dialog box. You can select a file by either double clicking on the file you want to open or clicking once to select the file and then pressing the **Open** button.

When you select **Save** from the **File** menu, the contents of the window will be saved into the file you just opened, regardless of the current directory.

How to Edit a Program

In the simplest case, you enter text by just typing. Press the **Enter** key to start the next line. You can erase text by pressing the **Backspace** key, which deletes the character directly behind the cursor (the small upward pointing arrow).

You can move the cursor by clicking the location in the **Editor window** that you wish to move the cursor to. At this point, you can insert text by typing, or delete text using the **Backspace** key.

Note that you can add blank lines at any point in the program by moving the cursor to the desired location and pressing the **Enter** key as often as required.

As the program expands beyond a single screen, use the scroll bars on the right-hand side to scroll the program up and down.

You can *select* a section of text using the mouse. Click down at the start of the text that you wish to select, and with the mouse button held down, drag the mouse to the end of the text that you wish selected and release the mouse button.

Once you have a *selection*, you can delete it by hitting the **Backspace** key or place a copy of it in the *paste buffer* by selecting **Copy** from the **Edit** menu. Once text is in the *paste buffer*, you can place a copy of the text at the location of the cursor by selecting **Paste** from the **File** menu.

When you have entered some portion of your program, you can *Paragraph* (indent) the program by pressing the **F2** function key or selecting **Paragraph file** from the **Edit** menu.

How to Print a Program

To print a copy of your program, select **Print** from the **File** menu. Note that you must have already have a printer selected.

How to Save a Program

To save a copy of your program, select **Save** from the **File** menu. If you loaded the program from the disk, this saves the contents of the window in the file that you opened. If the window is *unnamed*, a dialog box labelled **Save As** appears. You can now enter the file name for this program.

If you opened this window from a file and wish to save the contents of the window in a different file, select **Save As** from the **File** menu. This opens the **Save As** dialog box which gives you the opportunity to specify a different file name. The original file will not be changed.

How to Run a Program

To run a program, press the **F1** function key or select **Run** from the **Run** menu. OOT then attempts to compile the program. The **Status window** appears letting you know that OOT is compiling your program.

If errors are found in the program then compilation is unsuccessful. The **Error window** appears with the list of errors in the program and the location in the program where the first error occurred is highlighted. Correct the errors in the program and then run the program again.

If the program finished compiling successfully (several seconds the first time, a second or two subsequent times), execution starts. A **Run window** appears in the lower-right corner and all output from the program appears there. You also enter all input to the

program here. Note that the **Run window** must be active in order for keyboard input to be sent to the OOT program.

You can halt execution of the program by Selecting **Stop Execution** from the **Run** menu. At this point, you can print or close the **Run window**.

How to Print a Program's Output

To print the output of the **Run window**, make certain that the **Run window** is active and the select **Print** from the **File** menu. If you need to print out more than one screen's worth of output, see the section labelled *Input Redirection*.

How to Quit OOT

To finish using OOT, select **Quit** from the **File** menu.

Chapter 4

OOT for DOS

Installing OOT for DOS

Minimum System Requirements

DOS OOT requires an IBM PC compatible with a 386 processor and a minimum of 4 megabytes of memory. It also requires DOS 3.0 or better.

Installing Under DOS

To run DOS OOT, you require a minimum of a 386 system (386SX is completely acceptable) and 4 Mb of Random Access Memory (RAM). The more memory you have, the more DOS OOT can use, so extra memory is always helpful. You must also have 5 Mb of disk space available on your hard disk.

DOS OOT comes in two varieties. The commercial version requires the user to enter a registration number that is provided on the title page of this book (if purchased with software). The academic version requires a registration number that is provided separately. While the installation of the two is very similar, differences will be noted with the tag **Commercial** or **Academic** where a section applies only to one or the other.

Installation of DOS OOT is fairly straight forward. Place the installation diskette in drive A or B. Then type:

```
A:\INSTALL or B:\INSTALL
```

This starts the DOS OOT installer, which displays a window telling you what it intends to install.

Installer message telling you what it's going to install

Commercial: If this is the first time you've installed software, you'll get a notice reminding you to mail your registration card back.

Registration Card Reminder for Commercial Version

Commercial: Once these screens have passed, you'll continue on to the registration screen. The registration information screen asks you to enter some information about yourself and to enter your registration number. After you've installed the software once, you'll only be required to enter the registration number to install it again.

Registration Information Screen (Commercial)

Registration Information Screen Filled In (Commercial)

Commercial: The registration number must be filled in before you can continue the installation. You'll find the registration number on the lower-right corner of the title page of the **Turing Reference Manual**.

Academic: The first time you install the software, you'll be asked to enter the name of the organization (school or university). After the first time, this screen will not appear.

Registration Information Screen (Academic)

You can move between the different fields using the up and down arrow keys. When you are finished, move to the **OK** button and press **ENTER**. You will be given a chance to correct the information. If you decide to cancel the installation, move to the **Cancel** button using the up and down arrow keys and press **ENTER**.

Once you have entered the registration information, you will be asked to specify the installation directory (the directory in which DOS OOT is to be installed). Once you have entered the directory name, then move to the **OK** button and press **ENTER**. Selecting the **CANCEL** button will cancel the installation.

Object Oriented Turing Installation Directory	
Directory where you want Object Oriented Turing installed	
<input type="text" value="C:\OOT"/>	
< OK >	<Cancel>

Installation Directory Screen

If the disk you have chosen does not have enough space, a message will appear and allow you to select another disk. You can try to complete the installation even if there is theoretically not enough disk space. You might want to do this if you are installing over an old copy of DOS OOT

Once you have chosen the directory, the installation starts. You will get the following sequence of messages: **Installing DOS OOT**, **Verifying Software** and **Installing Support Files**. Each of these steps could take up to several minutes, so allow some time for the operation.

If there is a previous version of the software already on the disk, the installer gives you the option of whether you wish to install over top of the old version (and thus eliminate it) or create an additional one. Note that WinOOT and DOS OOT share the same OOT support files. Consequently if you have already installed WinOOT, there is no need to re-install the OOT Support Files again.

After the installation is complete, you will see the final message box indicating that the installation was successful.

Installation Complete Message

If there are any problems with the installation, please contact Technical Support for Holt Software Associates Inc. at (416) 978-8363.

De-installing Under DOS

To remove DOS OOT from the system, delete the directory that DOS OOT was installed in. Under DOS, this can be done using the **deltree** or **del/s** commands, depending on the version of DOS installed.

Using OOT for DOS

Starting DOS OOT

You can start using DOS OOT by either typing DOSOOT alone or with a filename (for example, DOSOOT PROGRAM.T). In the first instance, you get a blank window and no name is associated with your work until you try to save the file to the disk. The second form associates a filename with the work you will be doing and, if the file already exists, will initialize your OOT Environment to the contents of that file. There are other command line options for starting DOS OOT in specialized circumstances, which will be discussed later.

When you start up DOS OOT, you will see the **menu bar** at the top of the screen, the **status bar** at the bottom and an **alert box** in the center of the screen. The **alert box** gives the version number and the address of the publisher. Clicking the mouse or pressing any key will get rid of the alert box.

The Initial Screen With Start Up Message

Hitting any key or pressing the mouse button will get rid of the start up box.

The Initial Screen With No File Loaded

The bar at the top is the **menu bar**. Commands in the editor are usually given via the menus. The menu bar at the top of the program lists the menus. When a menu is selected, a menu bar is revealed showing the possible commands in that menu. There is the **File** menu, which mostly covers those commands that affect the entire file, the **Edit** menu which is used to make changes in the file itself, the **Search** menu which is used to both find and replace text in the file, and the **Run** menu which has commands pertaining to running the program.

At the bottom is the **status bar**. It always shows the line and column that the cursor is currently on and the name of the file being edited. The time of day appears in the upper left hand corner. The display of the time can be turned off by using the **noclock** startup option.

Selecting a Menu Item in DOS OOT

There are several ways to select menu items from a menu.

- Using the keyboard, press the **Alt** key and the first letter of the corresponding menu (i.e. **Alt-F** for the **File** menu, **Alt-E** for the **Edit** menu, etc.). The menu now appears and the top menu item in the menu is highlighted. Using the arrow keys, you move the highlight up and down. When the menu item you wish to select is highlighted, press the **Enter** key. The menu item is selected.
- As before, press the **Alt** key and the first letter of the corresponding menu (i.e. **Alt-F** for the **File** menu, **Alt-E** for the **Edit** menu, etc.). The menu now appears with the top menu item in the menu highlighted. You will note that most of the menu items have a letter highlighted in them. (For example the **N** in **New**, the **O** in **Open** and the **i** in **Pick** are all highlighted in the **File** menu.) By pressing the key that corresponds to the letter, the menu item is selected. Thus, by pressing **Alt-F** followed by **O**, the **Open** menu item would be selected from the **File** menu.
- Many of the menu items have a short cut listed beside the menu item name. For example, the **New** menu item has **Ctrl+N** listed beside it and the **Open** menu item has **Ctrl+O** and **F3** listed beside it. You can select a menu item without even selecting a menu by using the short cut. Thus pressing **Ctrl+O** in the editor would cause the **Open** menu item to be selected without actually displaying the menu.
- You can also use the mouse to select menu items. Press and hold the mouse down on the menu name. The menu now appears. Slide the mouse down with the button still pressed. As the mouse cursor goes over a menu item, the menu item is highlighted. Letting go of the mouse button will select the highlighted menu item.

If you want to select a different menu from the one that is current showing, you can:

- use the left and right arrow keys to display the next or previous menu, in which case the previous menu disappears and the new menu appears, or
- bring the mouse up to another menu name, in which case the previous menu disappears and the new menu appears.

If you decide not to select any menu at all, you can:

- press the **Esc** key, in which case all the menus disappear and no menu is selected, or
- move the mouse cursor off any of the menus and then release the mouse button, in which case all the menus disappear and no menu is selected.

Loading a Program in DOS OOT

To edit a program, you must select the **Open** menu item from the **File** menu. Using the keyboard, you can press **Alt-F** which displays the **File** menu:

The **New** menu item is highlighted. To select the **Open** menu item, you must use the arrow keys and press the down arrow until the **Open** menu item is highlighted. At that point, press **Enter**. This opens the **Open File** dialog box. As described above, this is only one way of selecting the **Open** menu item. You could also press **O** once the **File** menu was displayed, or you could have pressed **Ctrl+O** in the editor. You could also use the mouse to select the **Open** menu item.

The **Open File** dialog box has five sections.

- The top section contains the current directory name (C:\TURING\TEST*.*). In the current example, it is listing all the files in the directory as is indicated by the *.* at the end of the path name. You can specify it to list just the OOT files by pressing **F8** (i.e. the files that end in ".T"). If you press **F8** again, it goes back to listing all the files.
- The second section contains the buttons that you can choose. In this case, the **Tab** key will allow you to select another drive. The **F8** key will cause the dialog box to display only directories and files ending in ".T". The **F9** key sends you immediately back to the directory in which you started up DOS OOT. Clicking the mouse on any of the buttons has the same affect as pressing the key for that button.
- The third section is called the **File Name** section and allows you to directly type the name of a file or directory that you wish to load (or enter). If you type a file name and press **Enter**, that file will be opened. If you type a drive or directory name and press **Enter**, then the **Open File** dialog box is changed to display the contents of the new directory.
- This fourth section is called the **File** section and contains the names of the files and directories in the directory specified at the top of the dialog box. You maneuver through this section using the arrow keys, or using the mouse key to click on a file or directory name directly. When a file is highlighted, pressing **Enter** opens that file.

Double clicking on a file name also opens the file. If you select a directory instead of a file, the **Open File** dialog box is changed to display the contents of the new directory.

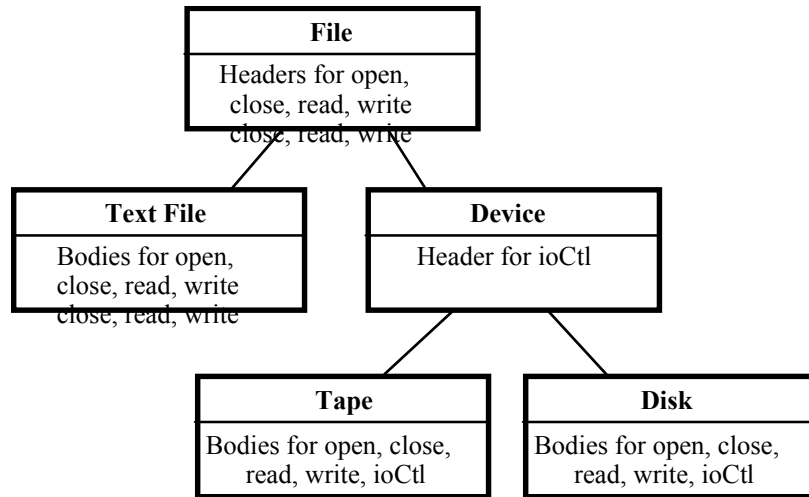
- The last section is called the **Drive** section and contains the drives that can be selected. Like the file section, you can use the arrow keys to select different drives. When the drive you want is highlighted, press **Enter** and the **Open File** dialog box will change to reflect the new drive and directory. Once again, double clicking on a drive will select the drive. To use the keyboard, once you are in the drive section (having pressed **Tab** to get there), you press **Tab** once again to get back to the **File** section.

You can always press **Esc** to cancel the **Open File** dialog box without opening a file.

Once you have selected a file, the file is opened and its contents appear in the **Editor window**. The status bar changes to reflect the name of the file just loaded. You also receive a message in the **message bar** (the section below the status bar) indicating how many lines were read in. At this point you are ready to start editing.

Editing a Program in DOS OOT

This is a sample screen with the program **EXAMPLE.T** just read in. Notice that the first and last lines on the screen extend beyond the right side of the screen. The right arrow on the right hand side is a reminder that the line goes beyond the right hand side of the screen. If you move the cursor off the right hand side of the screen, the screen scrolls horizontally. At that point, there will be left arrows reminding you that there is text to the left of the screen, as seen on the next page.



You can use the arrow keys to maneuver around the text, and type anything you like. By default, you start in **insert mode**, which means that anything you type is inserted in front of the character that the cursor is highlighting, instead of overwriting it. You can switch between insert and **overstrike mode** by pressing the **Insert** key on the keypad. Note that in insert mode, the cursor is a block, while in overstrike mode the cursor is an underline.

Saving a Program in DOS OOT

To save the program, you need only go into the **File** menu and select the **Save** menu item. To save the file with another name, select the **Save As...** menu item instead. The **Save As** dialog box appears (it will also appear if you select the **Save** menu item with a file that has not yet been named). The **Save As** dialog box looks like this.

The **Save As** dialog box is similar to the **Open File** dialog box, except that it has an extra section. The **File Name** section contains a place for you to enter the name of the file.

The cursor will be blinking in the first character of the **File Name** section. This is where you type in the name. If you wish to save the file in a different directory, you can specify the path name in the **File Name** section, or you can press **Tab** to get to the **File** section and select directories. Pressing **Tab** again moves you to the **Drive** section. A further **Tab** cycles around to the **File Name** section again. Pressing **Tab** repeatedly cycles between the **File Name** section, the **File** section and the **Drive** section.

Running a Program in DOS OOT

Select the **Run** menu by pressing **Alt-R** or using the mouse.

The **Run** menu item should already be highlighted. Press **Enter** to run the program. Once again note that you could have run the program by pressing **F1** or **Ctrl+R** without selecting a menu.

The program executes, and eventually you get a message telling you to press any key to continue. Upon doing so, you are returned to the program window. If there was an execution error, or if you terminated execution, the cursor is located on the line where the error or termination occurred. If it was an error, the line is also highlighted. Either way a message is displayed in the message area at the bottom of the screen.

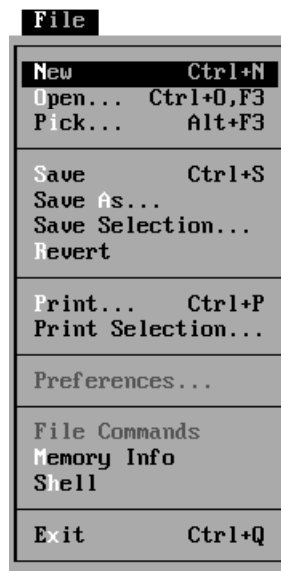
To Exit DOS OOT

To exit DOS OOT, go to the **File** menu (**Alt-F**) and select the **Exit** menu item. If the program has been modified and has not been saved, you get a chance to save it before quitting.

OOT for DOS Editor Reference

The Menu Commands:

The File Menu



New: This erases the file currently in memory and starts fresh. If the file has been changed and not saved, you get a chance to save it.

Open... This displays the **Open File** dialog box. The user can select the file to be opened from the **File** section.

Pick... This displays the **Pick File** dialog box which is a list of the last nine files opened. The user can select one of the files and it will be opened.

Save: Save the program in the window under the current file name. If there is no file name specified, treat the command as a **Save As**.

Save As... This displays the **Save As** dialog box. The user must enter a name for the file with the option of changing directory and drive.

Save Selection... This command is only active when there is a selection. If text has been selected, the user may save the selection using this command. This displays the **Save As** dialog box. Only the lines selected are saved to the file. This also

causes the selection to occupy complete lines (you can't save partial lines to the file).

Revert: This reloads the file from the disk, eliminating the changes made since the last save. The user is asked to confirm the change.

Print... This displays the **Print** dialog box. The user can select whether or not to print a header on the file and to print with line numbers.

Print Selection... This command is only active when there is a selection. If text has been selected, the user may print the selection using this command. This displays the **Print** dialog box. This also causes the selection to occupy complete lines (you can't print partial lines).

Preferences... This command is not available at this time and is always disabled.

File Command: This command is not available at this time and is always disabled.

Memory Info: This command displays a dialog box showing the current amount of memory available in the editor and on the PC.

Shell: This command allows the user temporary access to DOS. Type "**exit**" at the DOS prompt to return to DOS OOT. If the **noshell** startup option is selected, this menu item is always disabled.

Exit: This quits DOS OOT.

The Edit Menu

Undo: This command is not available at this time and is always disabled.

Cut: This command is only active when there is a selection. This copies the selection into the paste buffer and deletes the selection.

Copy: This command is only active when there is a selection. This copies the selection into the paste buffer. It does not change the selection.

Paste: This command places the contents of the paste buffer into the file. If there is no selection, it places it in front of the cursor. If there is a selection, it replaces the selection.

Clear: This command is only active when there is a selection. This deletes the selection.

Select All: This command selects the entire document.

Comment: This command is only active when there is a selection. This command places "%" in front of each line in the selection, making the lines into comments. This also causes the selection to occupy complete lines (you can't make a partial line a comment).

Uncomment: This command is only active when there is a selection. This command removes the "%" from the front of each line in the selection. You get an error

message if you try to uncomment a line that is not a comment. This also causes the selection to occupy complete lines.

The Search Menu

Find... This displays the **Find** dialog box. It allows the user to set the text to be searched for and to specify the direction and whether the find should "wrap" (i.e. continue looking from the beginning once it reaches the end of file).

Find Again: This command is only active when there is text that has been specified by the **Find** command. This command immediately looks for the next occurrence of the string specified by the **Find** command.

Find Error: This command is only active when there are errors in a Turing program. This jumps to the next error in the program. The error messages for that line will be displayed at the bottom of the program.

Change: This displays the Change dialog box. It allows you to do substitutions in the text. You can cycle through each occurrence of some text in the file, changing or leaving each occurrence. There is also an option to change every occurrence of some specified text in a file.

Goto Line: This displays a dialog box that allows you to enter a line number. Once the line number is entered, the program immediately jumps to that line.

The Mark Menu

The **Mark** menu contains the names of the first twenty procedures, functions, modules or classes in the currently opened file. Selecting a menu item causes the editor to scroll to the beginning of the construct selected. This feature can be used to help navigate through large files.

New procedures, functions, modules or classes are loaded into the **Mark** menu whenever the program is either loaded or paragraphed.

The Run Menu

Run: This command runs the program. If there are any compile-time errors, it immediately returns, jumping to the first line with an error in it. All errors are displayed just below the status bar. Any lines with errors are also highlighted.

Run with Args... This displays the **Run with Arguments** dialog box. This allows the user to specify the files that DOS OOT should read input from and write output to. Also allows the user to specify any run-time arguments to Turing.

Paragraph: This command paragraphs (indents) the program.

Fix Main Program: This brings up a dialog box allowing the user to specify the main program to be run whenever the user specified the **Run** command. This allows the user to edit a file containing a unit or an include file used by the main program and run the main program without having to load it into the editor each time.

Float Main Program: This command causes OOT to run the program in the editor when the **Run** command is given. This is used after the **Fix Main Program** command is given in order to allow the user to run different programs.

Fix Run Directory: This displays the **Set Directory** dialog box. This sets the directory from which the program will be run. If the program opens files for reading or writing, it will look for them here unless a path is specified.

Float Run Directory: This sets the execution directory to be the directory that the main program is saved in. If the main program has not been saved, then the program is run in the current directory (the directory you see when you select the **Open** command).

Reset Compiler: This resets the compiler. When the compiler is reset, the next time the program is run all the built in modules will be recompiled as well. This command can be used if the compiler is acting oddly in any way.

Compile to .EXE... This command is used to produce stand alone executables that allow the program to be run without having a copy of DOS OOT present. This command produces a ".EXE" file that combined with the DOS4GW.EXE that is provided with DOS OOT allows the program to be run on any 386 or better machine.

When the command is given, the **Save Executable** dialog box appears. This is similar to a **Save File** dialog box, except the default file name ends with .EXE. If the user wishes to rename the file, the file should always end in .EXE. After that, the **Compile Options** dialog box appears. This allows the user to select among options available for compiled programs. Once the options are selected, then the program is compiled. In order to execute the program, the user will have to leave DOS OOT.

The Help Menu

Note that for each of these commands, the cursor must be in or at the end of a keyword listed in the *Turing Reference Manual*.

Keyword Usage: This produces a one line summary at the bottom of the screen of the keyword selected.

Keyword Reference: This produces the entire *Turing Reference Manual* entry for the keyword selected. It appears in the **Editor window** and you can scroll up and down the entry using the arrow and page up/down keys. All other commands are disabled except the **Esc** key which returns the user to their program. The status bar changes from cyan to green when displaying a reference manual entry to further indicate that the user is not in the editor.

The Window Commands

There are a number of keypad commands that you can use to maneuver around the window, make selections and delete text.

Up/Down/Left/Right Arrows	Move up/down/left/right a character.
Home/End	Move to the beginning/end of the line.
Page Up/Down	Move up/down page (16 lines).
Ctrl+Left/Right Arrow	Move left/right a word.
Ctrl+Page Up/Page Down	Move to the top/bottom of the file.
Ctrl+Home/End	Move to the top/bottom of the screen.
Insert	This changes the current mode from Insert to Overstrike or Overstrike to Insert. The cursor is a block when in insert mode and an underscore in overstrike mode.
Ctrl+Y	Delete the current line and place it in the paste buffer.

Selecting Text

In order to be able to cut and paste blocks of text in the file, you need to be able to select that text. When text is selected, it is indicated by changing the background color of the text. By default, DOS OOT gives selected text a background color of blue.

Once the text is selected, it is possible to cut (remove) it, paste (place a copy of the text) it, print it, save it or even turn it into a set of comments.

To select text with the keyboard, you hold down the **Shift** key and use the cursor movement commands. Thus **Shift+Right Arrow** selects one character. **Shift+Down Arrow** selects one line of text. You can extend the selection by continuing to hold the shift key down while using the cursor movement commands. Note that you can select a page of a program at a time or even from the cursor to the beginning or end of a program by using combinations of the **Shift**, **Ctrl**, **Home**, **End**, **Page Up** and **Page Down** keys.

To select text using the mouse, move the mouse cursor to the point where you wish the selection to begin and press the mouse button. Holding the mouse button down, move the

mouse cursor to where you would like the selection to end and then release the mouse button. While holding the mouse button down, you can cause the screen to scroll by moving the mouse off the top, bottom, left or right of the screen.

Note that if there is a selection active and you do not hold the shift key down when using the cursor keys, or if you click the mouse anywhere else, the selection will become inactive (become unhighlighted).

Once the text is selected, you can operate on the text in a variety of ways. Commands that have an * work only on complete lines. When one of these commands is executed, the selection is extended to make complete lines.

Save Selection *	Saves the current selection.
Print Selection *	Prints the current selection.
Comment *	Makes the current selection a comment. This is a quick way of commenting out a section of code.
Uncomment *	Removes the comment symbol from the lines in the current selection. It will give an error if you try to uncomment a selection with lines that aren't comments.
Cut	Copies the selection to the paste buffer and deletes it from the file.
Copy	Copies the selection to the paste buffer but leaves the selection on the screen untouched.
Paste	This replaces the current selection with whatever is in the paste buffer.
Clear	This deletes the selection from the file.
Any keystroke	Any typing automatically replaces the selection with the keystroke typed. Pressing the backspace or the delete key deletes the current selection.

Mouse Movement

- The mouse can be used in a variety of ways in the file window. First, clicking in the file window moves the cursor to that point in the window.
- Double-clicking on a word selects the word.
- Triple-clicking on a line selects the line.
- Click-dragging the mouse will select text from the point the mouse was clicked on to the point that the button was released. When dragging the mouse, moving the mouse beyond the top of the screen causes the screen to scroll upward, while moving the mouse below the status bar causes it to scroll downwards.
- Clicking on the right-most column of the screen moves the cursor five spaces to the right. Holding the button down causes the cursor to continuously move to the right.

- Clicking on the left-most column of the screen when the screen has been scrolled to the right (i.e. when the left margin has left arrows in it) moves the cursor five spaces to the left. Holding the button down causes the cursor to continuously move to the left.

Dialog Boxes

The following applies to all dialog boxes:

The current default button has its brackets (the < >) in bright white.

In each button "<>", radio button "()" or checkbox"[]" there is often a character highlighted in bright white. Pressing either the letter (in dialog boxes where there is no text field such as the **Print** dialog box) or **Ctrl+letter** (in dialog boxes where there is a text field such as the **Find** dialog box) will cause a button to activate, a radio button to select that button, or a check box to be checked (or unchecked if it was already checked).

Pressing **Enter** activates the default button.

Pressing **Esc** cancels the dialog box without doing anything.

Pressing **Tab** cycles between the various objects in the dialog box.

You use the mouse to activate a button, select a radio button, or toggle a checkbox by clicking once on the item.

Some dialog boxes list all the files and directories in a particular directory. An individual file or directory can be selected either by double clicking on it with the mouse or by pressing **Enter** when the file or directory is highlighted. You can change which file or directory is highlighted by using the arrow keys. The "." directory moves the directory being displayed by the **File** dialog box to the parent directory.

Often buttons or files may be grayed-out in a dialog box. This means that the buttons or files are disabled and can not be selected.

Some dialog boxes list all the valid drives. The **File** dialog box can be switched to a particular drive by either double-clicking on the drive with the mouse, or by pressing **Enter** when the drive is highlighted. You can change the highlighted drive using the arrow keys.

The Open File Dialog Box

If there are more than 48 files and directories, then the dialog box will only display the first 48 files and directories. They will be displayed in columns of 12. There will be arrows on the right hand side to indicate that there are more files available to the right. (As you move to the right beyond the fourth column, arrows will appear on the left hand side to indicate that there are more files available to the left.)

In the **File Name** section, you use the keyboard to enter the name of the file. You may enter a full or partial path name. You can use **Ctrl+Y** to erase what has already been typed and use the standard cursor keys to maneuver in the text. You can enter only legal path name characters in the **File Name** section.

For the **File** and **Drive** section, the following applies.

Keyboard

- | | |
|---|--|
| Tab | Cycle between the File section and the Drive section. |
| F8 | Cycle between displaying only directories and files ending with ".T" and displaying all directories and files. |
| F9 | Switch to the directory in which you started DOS OOT. |
| A letter or number | Highlight the file or the drive that starts with that letter. |
| Esc | Cancel the dialog without selecting a file. |
| Enter (with a file highlighted) | Open that file. |
| Enter (with a directory highlighted) | Change the directory in the Open File dialog box to the selected directory. |
| Enter (with a drive highlighted) | Change the drive in the Open File dialog box to the selected drive. |
| Arrow Keys | Move the highlighted file up, down, left or right. |
| Home / End | Move the highlight to the first / last entry. |
| Page Up / Page Down | Move the highlight three columns to the left / right. |

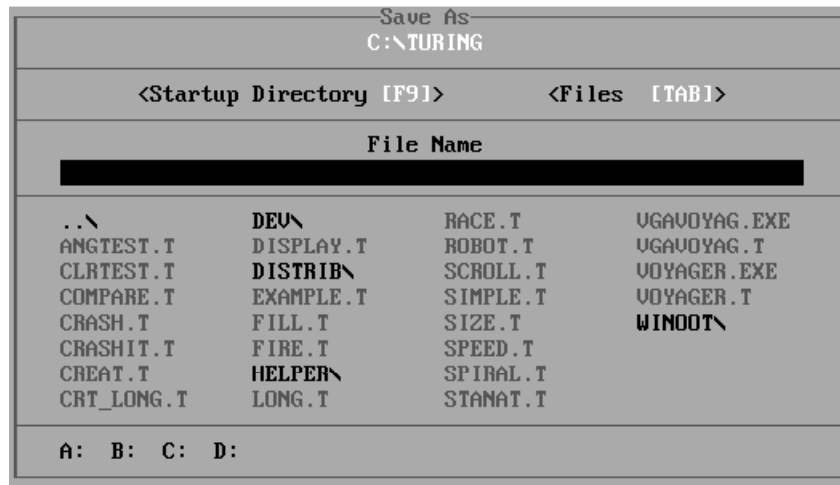
Mouse

- | | |
|--|--|
| Click on *.T (*.*) Files button | Cycle between displaying only directories and files ending with ".T" and displaying all directories and files. |
| Click on Startup Directory button | Switch to the directory in which you started DOS OOT. |
| Double click on a file | Open that file. |
| Double click on a directory | Change the directory in the Open File dialog to the selected directory. |

Double click on a drive

Change the drive in the **Open File** dialog to the selected drive.

The Save As Dialog Box



In the **Save As** dialog box, the file names (not the directory names) in the **File** section are grayed-out and may not be selected. If there are more than 48 files and directories, then the dialog box will only display the first 48 in columns of 12. There will be arrows on the right hand side to indicate that there are more files available to the right. (As you move to the right beyond the fourth column, arrows will appear on the left hand side to indicate that there are more files available to the left.)

In the **File Name** section, you use the keyboard to enter the name of the file. You may enter a full or partial path name. You can use **Ctrl+Y** to erase what has already been typed and use the standard cursor keys to maneuver in the text. You can enter only legal path name characters in the **File Name** section.

For the **File** and **Drive** section, the following applies.

Keyboard

Tab Cycle between the **File** section, the **Drive** section and the **File Name** section.

F9 Switch to the directory in which you started DOS OOT.

A letter or number Highlight the file or the drive that starts with that letter.

Esc Cancel the dialog without giving a file name.

Enter (with a directory highlighted)

Change the directory in the **Save As** dialog box to the selected directory.

Enter (with a drive highlighted)

Change the drive in the **Save As** dialog box to the selected drive.

Arrow Keys Move the highlighted file up, down, left or right.

Home / End Move the highlight to the first / last entry.

Page Up / Page Down

Move the highlight three columns to the left / right.

Mouse

Click on **Startup Directory** button

Switch to the directory in which you started DOS OOT.

Double click on a directory

Change the directory in the **Save As** dialog to the selected directory.

Double click on a drive

Change the drive in the **Save As** dialog to the selected drive.

The Print Dialog Box



This **Print** dialog box allows you to select the printing options. You can select whether the program is to be printed with a page header and whether line numbers are to be used when printing out the program.

Keyboard

Esc or **C** Cancel the dialog without printing.

O Start printing with the selected options.

H Toggle the Print Page Header option on and off.

L Toggle the Print Line Numbers option on and off.

Tab or **Arrow Keys**

Cycle the default button between **OK** and **Cancel**.

Enter Select the current button.

Mouse

Click on a check box

Toggle the option associated with that check box.

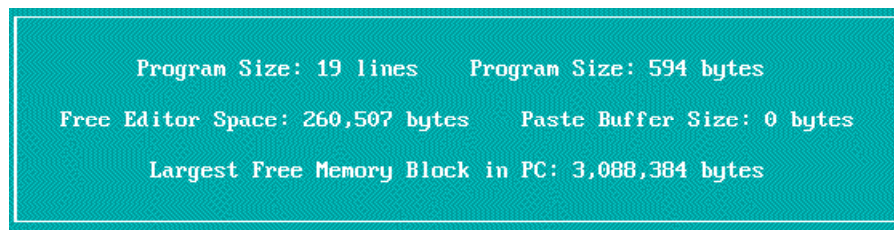
Click on **OK** button

Start printing with the selected options.

Click on **Cancel** button

Cancel the dialog without printing.

The Memory Info Alert Box



The **Memory Info** alert box requires no user input. It gives information about the amount of memory being used by the file in memory and the amount of free space available for use by DOS OOT. Pressing any key or clicking the mouse button will cause the Alert box to go away.

The Find Dialog Box

All

The **Find** dialog box is used to enter a **Find** string and then search for it. In general, it is expected that the user enters the string once and then uses **Find Again** (short cut **Ctrl+G**) to step through each occurrence found. Because the **Find** dialog box has a text field, you must use **Alt+keystroke** to toggle the buttons. You can use **Ctrl+Y** to erase what has already been typed and use the standard cursor keys to maneuver in the text.

Keyboard

Esc or **Alt+C** Cancel the dialog without searching.

Alt+O Find the next occurrence of the **Find** string.

Alt+B	Toggle whether to search for the next occurrence later in the text or the previous occurrence in the text.
Alt+W	Toggle wrap around search mode on and off. If you are searching downward in wrap around search mode and you reach the end of the file, find moves to the beginning of the file and continues the search. Likewise, if you are search backwards and you reach the beginning of the file, the search continues from the end of the file.
Enter	Select the default button (Cancel if no text has been entered, otherwise Find the next occurrence).

Mouse

Click on a check box	Toggle the option associated with that check box.
Click on OK button	Find the next occurrence of the Find string.
Click on Cancel button	Cancel the dialog without searching.

The Change Dialog Box



The **Change** dialog box is used to enter a **Find** string and then search for it. Once found, you can replace the **Find** string with the **Replace** string. Unlike other dialog boxes, the **Change** dialog box stays up on the screen until you dismiss it. Using it, you can easily change a small number of instances of a string, choosing which ones to leave intact and which ones to change as you see the instances in the file.

To use the **Change** dialog box, enter in a **Find** string, then press **Tab** to move the cursor to the **Replace** string. Once there, enter a **Replace** string. Once this is done, press **Enter** to find the first occurrence of the **Find** string. The screen will scroll and you will see the search string in the selection. If you want to replace this occurrence of the **Find** string with the **Replace** string, press **Enter** or click on the **Change, then Find** button. The selection will be replaced and the screen will automatically scroll to the next occurrence of the **Find** string. If you don't want to replace this occurrence, then press **Alt+F** or click on the **Find Next** button. This will cause the screen to scroll to the next occurrence without changing the selection.

To change the current selection without going to the next selection, press **Alt+H** or click on the **Change** button.

To change every occurrence of the **Find** string to the **Replace** string, press **Alt+A** or click on the **Change All** button.

When finished, press **Esc** or click the **Cancel** button in order to dismiss the dialog box.

For either text field, you can use **Ctrl+Y** to erase what has already been typed and use the standard cursor keys to maneuver in the text.

Keyboard

Esc	Cancel the dialog without searching.
Tab	Switch between editing the Find string and the Replace string.
Alt+B	Toggle whether to search for the next occurrence later in the text, or the previous occurrence in the text.
Alt+W	Toggle wrap around search mode on and off. If you are searching downward in wrap around search mode and you reach the end of the file, find moves to the beginning of the file and continues the search. Likewise, if you are search backwards and you reach the beginning of the file, the search continues from the end of the file.
Enter	Select the default button (Cancel if no text has been entered, Find Next if no text has yet been found, otherwise Change, then Find).
Alt+F	Find the next occurrence of the Find string.
Alt+C	Change the current selection to the Replace string and then find the next occurrence of the Find string.
Alt+H	Change the current selection to the Replace string.
Alt+A	Change every occurrence of the Find string to the Replace string.

Mouse

Click on Find Next button	Find the next occurrence of the Find string.
Click on Change, then Find button	Change the current selection to the replace string and then find the next occurrence of the Find string.
Click on Change button	Change the current selection to the Replace string.
Click on Change All button	Change every occurrence of the Find string to the Replace string.
Click on Cancel button	Cancel the dialog without searching.

Click on a check box

Toggle the option associated with that check box.

Click on a text field

Move the cursor to that text field and to the cursor position.

The Goto Line Box



The **Goto Line** dialog box allows you to jump directly to a particular line in the program. To use it, enter the line number you wish to jump to and press **Enter**. The text field will only allow you to enter numbers. Entering a number larger than the number of lines in the program will move you to the end of the program. You can use **Ctrl+Y** to erase what has already been typed and use the standard cursor keys to maneuver in the text field.

Keyboard

Esc or **Alt+C** Cancel the dialog without doing anything.

Alt+O Jump to the entered line in the program.

Tab or **Arrow Keys**

Cycle the default button between **OK** and **Cancel**.

Enter Select the default button.

Mouse

Click on **OK** button

Jump to the entered line in the program.

Click on **Cancel** button

Cancel the dialog without doing anything.

The Run with Arguments Dialog Box

The **Run with Arguments** dialog box is used to enter run-time arguments for the program about to be executed, and allow you to specify how you want the input and output to be redirected. To enter the arguments, type the arguments into the **Program Arguments** text field. To redirect the input or output, select the radio button (the "()") from the **Input From:** or the **Output To:** list. This can be done with either keyboard or mouse.

For the text field, you can use **Ctrl+Y** to erase what has already been typed and use the standard cursor keys to maneuver in the text.

Note that if you run the program without using **Run with Arguments**, then automatically no arguments are selected and the input and output are not redirected. The **Run with Arguments** dialog box also remembers the last settings so that you can choose between running with no arguments and running with a set of arguments and input and output redirection already set.

Keyboard

Esc or Alt+C	Cancel the dialog without doing anything.
Alt+R	Run the program with the option specified.
Tab	Cycle the default button between Run and Cancel .
Enter	Select the default button.
F3	Set input to come from the keyboard.
F4	Set input to come from a file. A dialog box identical in appearance to the Open File dialog box appears to allow you to select the input file.
F5	Set input to come from a file, but echo input from the file to the output device. See Input and Output Redirection . A dialog box identical in appearance to the Open File dialog box appears to allow you to select the input file.
F6	Set output to go to the screen.
F7	Set output to go to a file. A dialog box identical in appearance to the Save As dialog box appears to allow you to specify the output file.
F8	Set output to go to a file and the screen at the same time. See Input and Output Redirection later in this chapter. A dialog box identical in appearance to the Save As dialog box appears to allow you to specify the output file.
F9	Set output to go to the printer.
F10	Set output to go to the screen and a printer at the same time. See Input and Output Redirection later in this chapter.

Mouse

Click on Run button	Run the program with the option specified.
----------------------------	--

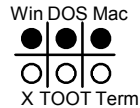
Click on **Cancel** button

Cancel the dialog without doing anything.

Click on a radio button

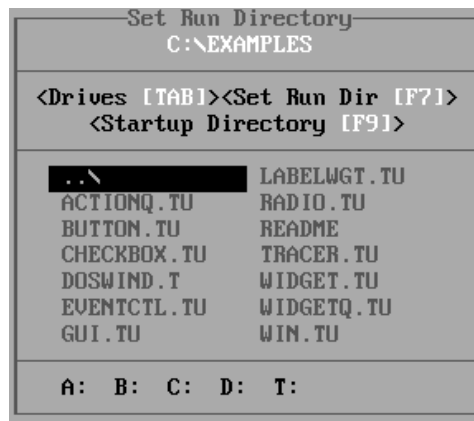
Set the option associated with that radio button.

The Set Main Program Dialog Box



The **Set Main Program** dialog box is used to select the program that you want to be run every time you select the **Run** command. As such, it is essentially identical to the **Open File** dialog box except that instead of opening a file, that file becomes the main program and will be run whenever the **Run** command is given. See the **Open File** dialog box section for more information.

The Set Run Directory Dialog Box



The **Set Run Directory** dialog box is used to select the directory in which you want your program to execute. Once it is set, any file I/O will use that directory as the base directory to read and write files. To use this dialog box, you simply change directories as you would with the **Open File** or **Save As** dialog boxes, and then select **Set Run Directory** when you are in the appropriate directory by either pressing **F7** or clicking on the **Set Run Directory** button with the mouse. All files are grayed out and may not be selected.

If there are more than 48 files and directories, then the dialog will only display the first 48 files and directories in columns of 12. There will be arrows on the right hand side to indicate that there are more files available to the right. (As you move to the right beyond

the fourth column, arrows will appear on the left hand side to indicate that there are more files available to the left.)

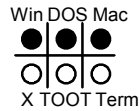
Keyboard

- Tab** Cycle between the **File** section and the **Drive** section.
- F7** Set the current directory displayed as the execution directory.
- F9** Switch to the directory in which you started DOS OOT.
- A letter or number
Highlight the file or the drive that starts with that letter.
- Esc** Cancel the dialog without selecting a file.
- Enter** (with a file highlighted)
No effect.
- Enter** (with a directory highlighted)
Change the directory in the **Open File** dialog to the selected directory.
- Enter** (with a drive highlighted)
Change the drive in the **Open File** dialog to the selected drive.
- Arrow Keys** Move the highlighted file up, down, left or right.
- Home / End** Move the highlight to the first / last entry.
- Page Up / Page Down**
Move the highlight three columns to the left / right.

Mouse

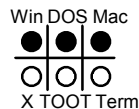
- Click on **Set Run Directory** button
Set the current directory displayed as the execution directory.
- Click on **Startup Directory** button
Switch to the directory in which you started DOS OOT.
- Double click on a file
No effect.
- Double click on a directory
Change the directory in the **Open File** dialog to the selected directory.
- Double click on a drive
Change the drive in the **Open File** dialog to the selected drive.

The Save Executable Dialog Box



The **Save Executable** dialog box allows the user to specify the name of a the compiled program when compiling a file. The file name for the executable should always in ".EXE" in order so that the DOS will be able to execute the program when the name is typed. Otherwise this dialog box is identical to the **Save As** dialog box. See the **Save As** dialog box for more information.

The Compile Options Dialog Box



The Compile Options dialog box allows the user to select the certain conditions on how the compiled program will execute from DOS.

When Ctrl+Break is disabled, the user of the compiled program will not be able to stop execution of the program except by turning the machine off. Ctrl+Break, Ctrl+C and Ctrl+B will have no effect.

When No Return to DOS Message is set, when the program stops execution the user will be immediately returned to the DOS prompt. If the program was in graphics mode, the display will be set back to text mode erasing the screen. When this option is not selected, when the program finishes execution, a message is displayed asking the user to press a key to return to DOS.

When "No Erase before Execution" is set, the program will not erase the screen before starting execution or after stopping execution. Note that if you switch to graphics mode, the screen will be erased anyway. This option is most useful for DOS command line utilities. Programs compiled with DOS OOT will however display the DOS 4G/W (the DOS extender) banner before execution.

Keyboard

Esc or C Cancel the dialog without compiling.

O Start printing with the selected options.

Tab or Arrow Keys

Cycle the default button between **OK** and **Cancel**.

B	Toggle the Ctrl+Break Disabled check box.
R	Toggle the No Return to DOS Message check box.
E	Toggle the No Erase Before Execution check box.
Enter	Select the current button.

Mouse

Click on a check box	Toggle the option associated with that check box.
Click on OK button	Compile the program with the options selected.
Click on Cancel button	Cancel the dialog without compiling.

The Pick File

DOS OOT creates a file call **TURING.PCK** in the directory from which it was started. This contains the names of the last nine files loaded into DOS OOT for use with the **Pick...** menu item. Eliminating the file has no harmful consequences except that the **Pick** command will not remember any previous files loaded in.

For single user systems, it may be convenient to have the **TURING.PCK** file stored only in one location (the directory that stores **TURING.EXE**) and have this file used no matter where you start up DOS OOT. You can change DOS OOT to have this behavior by using the **centralpickfile** start up option. For more about start up options, see the next section.

Start Up Options

There are three places you can specify startup options. These are:

- in the **TURING.CFG** file in the directory where **DOSOOT.EXE** is found.,
- in the **TURING.CFG** file where the user started DOS OOT,
- and in the actual command line used in starting up DOS OOT.

The best way to set a standard set of options for all session of DOS OOT is to put all the options in the file **TURING.CFG** in the same directory as **TURING.EXE**. That way, all sessions will have the same startup options. You can then customize those options for individual sessions by making your own **TURING.CFG** file in the directory in which you start the session.

To specify a startup option in a **TURING.CFG** file, the file should list the options, one per line. It is not necessary to put a "-" or a "/" in front of the option names. For example, a **TURING.CFG** file could contain the following line:

```
printer=LPT2
```

This would select **LPT2** as the default printer. In the command line, options must be preceded with either a "-" or a "/". For example, you could start DOS OOT up with:

```
dosoot /alertclr=brown
```

Here is the complete list of available startup options. Options that have a * beside them are the default (i.e. are selected if no option is given).

backup *	Automatically save the old file with ".BAK" extension.
nobackup	Don't save the old file.
nopara	Disable the Paragraphing menu item.
noshell	Disable the Shell menu item and the system procedure.
formfeed *	Print a form feed at the end of each print job.
noformfeed	Don't do a form feed at the end of each print job.
tabs *	Convert spaces to tabs when printing and saving.
notabs	Don't convert spaces to tabs when printing and saving.
header *	Print a header at the top of each print job with the date and file name.
noheader	Don't print a header at the top of each print job.
overstrike	Start in overstrike mode.
insert *	Start in insert mode.
mono	Force system to use monochrome display adapter.
graphics	Force DOS OOT to use a graphics display adapter.
checkclr *	Check color number for legality on draw... commands.
nocheckclr	Don't check color number for legality on draw... commands.
bw	Assume a black and white monitor.
clr	Assume a color monitor.
allprog *	In file dialogs, display all files by default.
tprog	In file dialogs, display only files ending with ".T" by default.
arrowcursor *	Use an arrow mouse cursor in the editor.

noarrowcursor	Use standard block mouse cursor in editor. Use this option if cursor appears as a group of Greek letters.
laserjet	Use HP LaserJet control codes when printing. Use this option if you are using an HP LaserJet that is <i>not</i> configured for Epson emulation.
nolaserjet *	Use standard printer end of line code.
clock *	Display the time in the upper right corner of the screen.
noclock	Don't display the time in the upper right corner of the screen.
logo *	Display the alert at startup that gives the version number.
nologo	Don't display the alert at startup that gives the version number.
specialclear	On certain systems, the cls command doesn't work in MCGA mode. Set this option to get it to work.
savepick *	Save the pick file when using DOS OOT.
nosavepick	Don't save a TURING.PCK file when using DOS OOT.
mouse *	Allow use of the mouse in the editor.
nomouse	Don't allow use of the mouse in the editor.
useb	Always display drive B, even if there's only one drive.
nouseb	Display drive B, only if there are two drives.
centralpickfile	Use and save the TURING.PCK file found in the same directory as TURING.EXE .
nocentralpickfile *	Use and save the TURING.PCK file found in the directory from which DOS OOT was started up.
nosound	Stop DOS OOT from producing any sound either in the editor or using the sound commands in the language.
resetmouse	Use this option if the mouse doesn't seem to be working. Forces the mouse driver to do a hard reset.
cga	Force DOS OOT to use CGA as the default graphics mode (as it does in Turing). Use this option for better compatibility with Turing programs.
vesa	Force the DOS OOT graphics library to use VESA graphics commands for SuperVGA graphics modes. Normally DOS OOT correctly detects the graphics card in the machine, however in some systems, it can fail (for example, the ATI Mach64 graphics card). Use this option to get them to work correctly.

novesa*	Autodetect the graphics adapter and use the appropriate SuperVGA graphics commands where applicable.
printermargin=<i>n</i>	Set the left hand margin on printouts to start printing <i>n</i> spaces to the right. By default, <i>n</i> = 0.
linesperpage=<i>n</i>	Set DOS OOT to print a form feed every <i>n</i> lines. By default, <i>n</i> = 0 which corresponds to not doing any form feeds.
stack=<i>n</i>	Set DOS OOT to allocate <i>n</i> kilobytes for the stack during execution. By default <i>n</i> = 32.
line=<i>n</i>	Set DOS OOT to have a line buffer size of <i>n</i> kilobytes. By default, <i>n</i> = 64.
maxfiles=<i>n</i>	Set the maximum number of files that DOS OOT can read in a directory. By default <i>n</i> = 150.
printer=<i>name</i>	The name of the printer that DOS OOT will send printouts to (default: LPT1).
fontdir=<i>name</i>	The name of the directory where the MS Windows fonts are stored (for use by the Font module) (default: C:\WINDOWS\SYSTEM).
startup=<i>name</i>	The directory that the user will be in when DOS OOT starts up (default: The current directory).

Note that in the following, colors (*clr*) can be from 0 to 15, or one of **black**, **blue**, **green**, **cyan**, **red**, **magenta**, **purple**, **brown**, **white**, **grey**, **gray**, **brightblue**, **lightblue**, **brightgreen**, **lightgreen**, **brightcyan**, **lightcyan**, **brightred**, **lightred**, **brightmagenta**, **lightmagenta**, **brightpurple**, **lightpurple**, **yellow**, **brightwhite** or **lightwhite**.

Background colors (*bkclr*), can be from 0 to 7, or one of **black**, **blue**, **green**, **cyan**, **red**, **magenta**, **purple**, **brown** and **white**.

menuclr=<i>clr</i>	The color of text of menu items (default: black).
menudisabledclr=<i>clr</i>	The color of text of disabled menu items (default: gray).
menuintenseclr=<i>clr</i>	The color of text of intense letters in the menu (default: brightwhite).
menubklclr=<i>bkclr</i>	The color of the background in the menus (default: white).
textclr=<i>clr</i>	The program text color (default: white).
texthiliteclr=<i>clr</i>	The program highlighted text color (default: brightwhite).
textbkclr=<i>bkclr</i>	The background color of the program (default: black).
textslectbkclr=<i>bkclr</i>	The color of the background in selected text (default: blue).
statusclr=<i>clr</i>	The color of the text in the status line (default: black).
statusbkclr=<i>bkclr</i>	The color of the background of the status line (default: cyan).

messageclr=clr	The color of editor messages (default: yellow).
messagebkclr=bkclr	The background color of the editor messages (default: black).
dlgclr=clr	The color of text in dialog boxes (default: black).
dlgintenseclr=clr	The color of intense text in dialog boxes (default: brightwhite).
dlgbkclr=bkclr	The background color of dialog boxes (default: white).
dlgdisabledclr=clr	The color of disabled text items in dialog boxes (default: gray).
dlgborderclr=clr	The color of the border in dialog boxes (default: brightwhite).
alertclr=clr	The color of the text in alert boxes (default: black).
alertbkclr=bkclr	The background color in alert boxes (default: red).
progrerrclr=clr	The color of DOS OOT error messages (default: brightred).
progrerbkclr=bkclr	The background color of the DOS OOT error messages (default: black).

Compilation to Executable File

DOS OOT has the ability to compile files to produce a standalone executable (a file that does not need DOS OOT in order to run). To produce the stand alone executable, the user should select **Compile to .EXE** from the **Run** menu, supply a filename that ends in ".EXE" and compile with the desired compile options (explained previously in **The Compile Options Dialog Box** section).

The executables produced are yours alone to distribute or sell as the user sees fit. No royalties or restrictions apply.

The compiler works by compiling the program to pseudo-code (a sort of generic assembly language) and then attaching it to the DOS OOT executor. The DOS OOT executor is fairly large, so users will find that all generated executables are about 400KB in size (the size of the executor) and then grow slowly after that.

The DOS OOT executor is a Extended DOS application. This means that a DOS extender must be available to run the program. The DOS extender is called DOS/4GW and must be included with the executable file. Thus to distribute your program, you must make a copy of both your executable program, any data files it may use and **DOS4GW.EXE**. **DOS4GW.EXE** can be found in the same directory as DOS OOT and may be freely copied from there. Note that when running the program, **DOS4GW.EXE** must be either in the current directory or in the DOS path.

Note that compilation of a program also compiles all the units (modules and classes) and include files that the program uses. It is not necessary to distribute any source code

with the executable file. Data files (files opened using the **open** statement) are not compiled in and must be included separately.

Input and Output Redirection

DOS OOT has the capability of redirecting program input and output. This enables a user to write a program that normally reads from the keyboard and writes to the screen to read input from a file and send output to a file at run-time. This can be used for a variety of purposes such as testing programs with certain input from a file or saving a copy of output to a file for later use.

The method for redirecting input or output is to use the **Run with Arguments** menu item. When the dialog box comes up, you are given the opportunity to redirect input to come from the **keyboard**, **file** or **file with echo**. Likewise, you can redirect output to go to the **screen**, **file**, **screen and file**, **printer** or **screen and printer**.

Because a file can hold only textual information, using graphics when sending output to a file causes a run-time error.

The exact meaning of the redirection command is given below:

Input from:

file	Read input from a specified file instead of the keyboard. Do not echo the input to the screen.
file with echo	Read input from a specified file instead of the keyboard. Echo the input to the screen.

Output to:

file	Send output to a specified file instead of the screen. Do not display any output on the screen.
screen and file	Send output to the screen and a specified file.
printer	Send output to the printer instead of the screen. Do not display any output on the screen.
screen and printer	Send output to the screen and the printer.

Summary

Input from keyboard / Output to screen

Reads input from keyboard. Sends output to the screen.

Input from infile / Output to screen

Reads input from **infile**. Sends output to the screen. Input from **infile** is not seen on the screen.

Input from infile with echo / Output to screen

Reads input from **infile**. Sends output to the screen. Input from **infile** is seen on the screen.

Input from keyboard / Output to outfile

Reads input from keyboard. Sends output to **outfile**. Input from the keyboard appears on the screen and not in **outfile**.

Input from infile / Output to outfile

Reads input from **infile**. Sends output to **outfile**. Input from **infile** is not seen on the screen or in **outfile**. The screen is blank while the program is running because all output is going to **outfile**.

Input from infile with echo / Output to outfile

Reads input from **infile**. Sends output to **outfile**. Input from **infile** is seen in **outfile**. The screen is blank while the program is running because all output is going to **outfile**.

Input from keyboard / Output to screen and outfile

Reads input from keyboard. Sends output to both screen and **outfile**. Input from keyboard appears on both the screen and in **outfile**.

Input from infile / Output to screen and outfile

Reads input from **infile**. Sends output to both screen and **outfile**. Input from **infile** is not seen on the screen or in **outfile**.

Input from infile / Output to screen and outfile

Reads input from **infile**. Sends output to both screen and **outfile**. Input from **infile** appears on both the screen and in **outfile**.

Input from keyboard / Output to the printer

Reads input from keyboard. Sends output to the **printer**. Input from the keyboard appears on the screen and not on the **printer**.

Input from infile / Output to the printer

Reads input from **infile**. Sends output to the **printer**. Input from **infile** is not seen on the screen or on the **printer**. The screen is blank while the program is running because all output is going to the **printer**.

Input from infile with echo / Output to the printer

Reads input from **infile**. Sends output to the **printer**. Input from **infile** is seen on the **printer**. The screen is blank while the program is running because all output is going to the **printer**.

Input from keyboard / Output to screen and the printer

Reads input from keyboard. Sends output to both screen and the **printer**.
Input from keyboard appears on both the screen and on the **printer**.

Input from infile / Output to screen and the printer

Reads input from **infile**. Sends output to both screen and the **printer**. Input from **infile** is not seen on the screen or on the **printer**.

Input from infile / Output to screen and the printer

Reads input from **infile**. Sends output to both screen and the **printer**. Input from **infile** appears on both the screen and on the **printer**.

Chapter 5

The GUI Module

Introduction

Since the introduction of the Macintosh, graphical user interfaces (GUI) have been becoming more and more common. Most commercial programs written for either the Macintosh or Microsoft Windows make use of GUI elements to make their program easier to use.

For some time, students have been requesting methods of putting GUI elements such as buttons, check boxes, radio buttons, etc, into their Turing programs. With the Fall 1998 release of OOT, we introduced a new set of predefined subprograms that allow students to add numerous GUI elements to their programs quickly and easily. These subprograms allow students to create: buttons, check boxes, radio buttons, sliders, scroll bars, picture buttons, radio picture buttons, text fields, lines, text labels, and frames.

The entire GUI Library is written in Turing and the source is included with the OOT distribution. The GUI library is completely procedure-oriented. This means that it is not necessary to know object-oriented programming or concepts in order to be able to use the library. Advanced students are welcome (in fact, encouraged) to look at the programs as an example of a large project written in Turing. We hope that there will be enterprising students who will be inspired to add new widgets to the library and encourage those who do so to submit them to Holt Software for possible inclusion into the next version of the library.

Here is a window with a few widgets.

Some GUI Widgets (Output of Example.dem)

The GUI library is usable by students who understand the concept of subprograms. In order to use the GUI library, students must write procedures (although they may be as simple as the student desires). We therefore suggest that teachers introduce students to the GUI library in a Grade 11 computer science course.

Note: Object Oriented Turing has not changed. It is not a visual building language. Students wishing to use the GUI library will be writing programs to create and use these

GUI element, not spending their time visually building user interfaces (which may be fun, but teaches very little). In keeping with the tradition of Turing, the more the students learn about computer science, the more interesting their programs will be, GUI or no GUI!

Terms

This document will use the term "Turing" to refer to the Turing and Object Oriented Turing. Thus when we speak of Turing programs, this will mean programs written in Turing or Object Oriented Turing.

The term "Widget" means any graphical user interface element. A button, a check box or radio button are all examples of widgets.

The term "Event" means either a keystroke or a mouse button being pressed in a window.

Example Programs

All example programs shown here are located in the `%oot/examples/GUI` directory. (In other words, start in the same directory as WinOOT, move to the examples folder and then the GUI folder.) All the available GUI widgets have example programs to demonstrate their use. Examine the file **README.TXT** for a description of the files.

General Principles of the GUI Library

Here are some general instructions for the use of the GUI library. Read this section before looking at the specifics of various routines.

- All the subprograms for the GUI library are placed in a module called GUI. To call any of the subprograms, preface the name of the subprogram with "GUI.". For example, the *CreateLabel* subprogram would be called using *GUI.CreateLabel*.
- In general, most widgets have a *Create* subprogram. For example, buttons have a *CreateButton* subprogram, radio buttons have a *CreateRadioButton* subprogram, and so on. The *Create* subprogram takes as parameters things such as the location, the size of the GUI element, and the name of a procedure to be called when the widget is clicked. This procedure must be declared before the call to the *Create* subprogram.
- For most widgets, there are two forms of the *Create* subprogram. The *Create* subprogram and the *CreateFull* subprogram. The difference between the two is that the *CreateFull* subprogram allows the user to define more parameters that are otherwise set to default values. For example, the *GUI.CreateButton* procedure allows the user to specify the x and y location of the button, the width of the button, the text that appears in the button, and the procedure to call when the button is clicked. The *GUI.CreateButtonFull* routine specifies those and also allows the user to specify the height of the button (otherwise set to a height that will fit the label), a short cut keyboard character that allows the user to "press" the button using a keyboard and a

parameter to allow the user to specify if this button is the "default" button (the one "pressed" if the user presses the Enter key).

- All *Create* subprograms return an integer. This number is the ID of the widget that has been created. You need to use this ID if you want to do anything to the widget later, such as move it, change its size, hide it, and so on. Most simple programs can safely ignore the widget ID, although they will need to handle the return value from the function.
- After all the widgets have been created, the program must repeatedly call *GUI.ProcessEvent* until the function returns **true**.

```
% Now process events until the user aborts the program.  
loop  
    exit when GUI.ProcessEvent  
end loop
```

GUI.ProcessEvent checks for user input from the mouse or the keyboard and then checks to see if the user has clicked on a widget. If the user has, then it responds appropriately (toggling the check box, pressing the button, etc.) and then if appropriate, calls the procedure the user supplied in the *Create* subprogram. *GUI.ProcessEvent* returns **true** when the *GUI.Quit* has been called, otherwise it returns **false**.

- When a program is finished execution (for example if the user selected "Quit" or "Exit" from the file menu), it should call the *GUI.Quit* procedure. This will cause the *GUI.ProcessEvent* loop to exit. The program should have any clean up code placed after the **end loop**.

Here is a very simple example of a program that puts "Hello" every time a button is pressed.

```
% The "Hello" program.  
import GUI in "%oot/lib/GUI"  
  
View.Set ("graphics:200;200") % Shrink the run window  
  
% The procedure called when the button is pushed.  
procedure PutHello  
    put "Hello"  
end PutHello  
  
% Create the button. The number returned is the ID number of the button.  
var b : int := GUI.CreateButton (100, 100, 0, "Say Hello", PutHello)  
  
% Now process events until the user aborts the program.  
loop  
    exit when GUI.ProcessEvent  
end loop
```

Here is the output window after the user has pressed the button once.

Output of Hello.dem

Active and Passive Widgets

Widgets come in two forms. Active widgets are ones that respond to keystrokes and button clicks. Passive widgets do not respond to anything. Examples of passive widgets are lines, frames, labels, labelled frames and pictures. Passive widgets are generally used to organize the output window.

Here is an example of a small program that show some passive widgets.

```
% The "passive" program
% This demonstrates some of the passive widgets such as:
% Lines, Frames, Labelled Frames, Labels and Pictures.
import GUI in "%oot/lib/GUI"

% We'll need a picture for our Picture widget. Most likely
% you would normally have it saved in an external file and
% use Pic.FileNew to read it into a picture. For the example
% program we'll construct it by hand.
Draw.FillOval (50, 50, 50, 50, blue)
Draw.FillBox (17, 17, 83, 83, brightred)
Draw.FillStar (17, 17, 83, 83, brightgreen)
Draw.FillMapleLeaf (37, 37, 63, 63, brightpurple)
var pic := Pic.New (0, 0, 100, 100)

View.Set ("graphics:310;335")

#if _DOS_ then
  Draw.Cls
#else
  % The background must be gray for indented and exdented
  % items to be visible.
  GUI.SetBackgroundColor (gray)
#end if

% Now place the widgets.
% Three lines of the different types with labels
var line1 := GUI.CreateLine (70, 10, maxx - 10, 10, GUI.LINE)
var label1 := GUI.CreateLabelFull (60, 10, "Line", 0, 0,
  GUI.RIGHT + GUI.MIDDLE, 0)
var line2 := GUI.CreateLine (70, 30, maxx - 10, 30, GUI.INDENT)
var label2 := GUI.CreateLabelFull (60, 30, "Indent", 0, 0,
  GUI.RIGHT + GUI.MIDDLE, 0)
var line3 := GUI.CreateLine (70, 50, maxx - 10, 50, GUI.EXDENT)
```



```

var label3 := GUI.CreateLabelFull (60, 50, "Exdent", 0, 0,
    GUI.RIGHT + GUI.MIDDLE, 0)

% Now place the frames
var frame1 := GUI.CreateFrame (10, 70, 100, 120, GUI.LINE)
var label4 := GUI.CreateLabelFull (10, 70, "Line", 90, 50,
    GUI.CENTER + GUI.MIDDLE, 0)
var frame2 := GUI.CreateFrame (110, 70, 200, 120, GUI.INDENT)
var label5 := GUI.CreateLabelFull (110, 70, "Indent", 90, 50,
    GUI.CENTER + GUI.MIDDLE, 0)
var frame3 := GUI.CreateFrame (210, 70, 300, 120, GUI.EXDENT)
var label6 := GUI.CreateLabelFull (210, 70, "Exdent", 90, 50,
    GUI.CENTER + GUI.MIDDLE, 0)

% Now place the labelled frames
var frame4 := GUI.CreateLabelledFrame (10, 140, 100, 190, GUI.LINE, "Line")
var frame5 := GUI.CreateLabelledFrame (110, 140, 200, 190, GUI.INDENT,
    "Indent")
var frame6 := GUI.CreateLabelledFrame (210, 140, 300, 190, GUI.EXDENT,
    "Exdent")

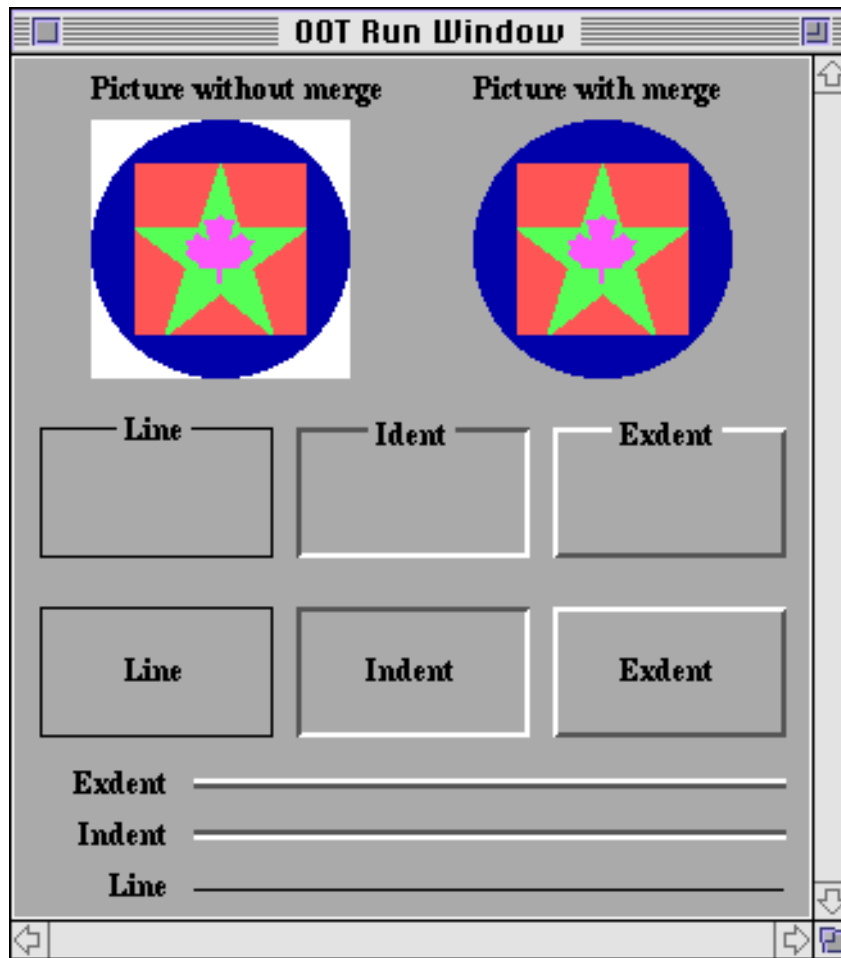
% Place the picture
var label7 := GUI.CreateLabel (30, 315, "Picture without merge")
var pic1 := GUI.CreatePicture (30, 210, pic, false)

#if not _DOS_ then
    var label8 := GUI.CreateLabel (maxx - 130, 315, "Picture with merge")
    var pic2 := GUI.CreatePicture (maxx - 130, 210, pic, true)
#end if

% This loop doesn't do much since none of the widgets have any actions.
loop
    exit when GUI.ProcessEvent
end loop

```

Here is the output window from the program with some labels, a line, a picture, and a labelled frame.



Output of Passive.dem

When an active widget is initialized, usually an *action procedure* must be specified. This is the name of a procedure that will be called when the widget is selected. For example, in the *Hello* program, the *PutHello* procedure was specified as the *action procedure* of the button. Whenever the button was pressed, the *PutHello* procedure was called.

Some *action procedures* have arguments. For example, the *action procedure* for a slider has a parameter of the current value. This allows the procedure to use the current value without having to call a GUI subprogram to get the current slider value.

Keyboard Shortcuts

Several types of widgets can have “shortcuts”. A shortcut is simply a keystroke that has the same effect as clicking on the widget. When you specify a shortcut to a widget in the *CreateFull* procedure for the widget, you must specify a single character. The easiest way to do this is to use the *chr* function with the ASCII value of the character to be used as the shortcut. You can also specify control characters using the “^” notation. For example, the character Ctrl+F can be expressed as “^F” in Turing.

The following characters cannot be used as shortcuts because the OOT environment uses them for various purposes (stopping or rerunning programs, and so on.): Ctrl+C, Ctrl+D, Ctrl+Z, F1, F11 and F12.

Background Color

It is common for windows to have a different background color from the standard white (or black under DOS). To change the background color of a window (or the screen under DOS), use the *GUI.SetBackgroundColor* procedure. This procedure takes one parameter, the new background color. It redraws the window in the background color and then redraws all the widgets. It also notifies the widgets about the new background color so that when the widget is erased or moved, the location of the widget is filled with the new background color instead of white (or black under DOS).

Note that Microsoft Windows dialog boxes often have a background color of *gray*. In order to simulate that, you should give the command *GUI.SetBackgroundColor (gray)* before creating widgets.

Several widgets (Canvas, Frame, Labelled Frame, Text Field and Text Box) can have borders of either type INDENT or EXDENT. These borders give a sort of 3-D appearance to the widget. However, they require that the background be set to *gray*.

Here is an example of a small program that creates a Canvas with a 3-D appearance and then draws circles in the corner.

```
% The "Canvas1" program.
% Create a canvas and draw four circles on it.
import GUI in "%oot/lib/GUI"

View.Set ("graphics:200;200")

#if not _DOS_ then
    % Necessary for a 3-D look for the canvas
    GUI.SetBackgroundColor (gray)
#end if

% This procedure is needed as an argument to CreateCanvasFull.
procedure DoNothing (mx, my : int)
end DoNothing

% Create a label for the canvas. We could use CreateLabelFull for more
% precise alignment.
```

```

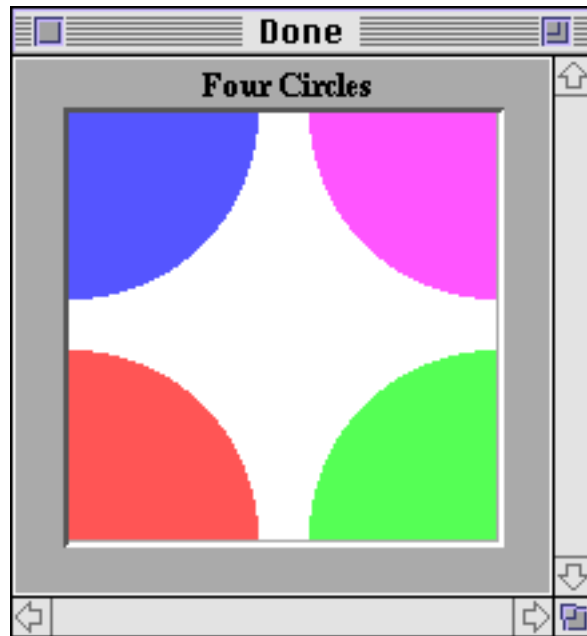
var label1 := GUI.CreateLabel (70, 182, "Four Circles")

% Create the canvas. We need to use CreateCanvasFull in order to
% specify the type of border.
var canvas := GUI.CreateCanvasFull (20, 20, 160, 160, GUI.INDENT,
    DoNothing, DoNothing, DoNothing)

% Draw the four ovals. Notice that they don't extend off the canvas
% and the co-ordinates they use are relative to the canvas, not the window.
const radius := 70 % Half the width - 10
GUI.DrawFillOval (canvas, 0, 0, radius, radius, brightred)
GUI.DrawFillOval (canvas, 160, 0, radius, radius, brightgreen)
GUI.DrawFillOval (canvas, 0, 160, radius, radius, brightblue)
GUI.DrawFillOval (canvas, 160, 160, radius, radius, brightpurple)

```

Here is the output window.



Output of Canvas1.dem

Widget Sizes

The size that you specify a widget to be is not necessarily the actual size that the widget will appear. In fact for many widgets, you can specify a width and height of 0 for the widget and let the initializer decide how large the widget should be. Another example is with check boxes, where if you specify the check box to be right justified, the x and y coordinates indicate the lower-*right* corner instead of the lower-left corner as usual. This

means that you may have to do some experimentation to determine where you want the widgets to be placed. Read the page on each subprogram that you use to find out exactly what you are specifying with the *x*, *y*, *width* and *height* parameters.

If you are trying to align widgets together (for example aligning scroll bars with a canvas), use the *GUI.GetX*, *GUI.GetY*, *GUI.GetWidth*, and *GUI.GetHeight* functions to determine the size of the object.

Positioning Text Labels (Aligning Labels with Widgets)

It is very common to want to align text labels with widgets on the screen. There are a few tips and tricks to doing so successfully. To align a text label with a widget, it is simply a matter of using the *GUI.CreateLabelFull* function with the appropriate *x*, *y*, *width*, *height* and *alignment* arguments.

If you are left or right aligning a label, then generally you will want the *x* coordinate to specify the edge to be aligned from and the *width* parameter should be set to 0. Similarly, if you are top or bottom aligning a label, then the *y* coordinate should specify the edge to be aligned from and the *height* parameter should be set to 0.

To align a widget horizontally with a widget, choose *GUI.CENTER* for the horizontal alignment and use the *x* coordinate and *width* of the widget as the label's *x* coordinate and *width*. You can get the *x* coordinate and width of a widget using *GUI.GetX* and *GUI.GetWidth*.

Likewise, to align a widget vertically with a widget, choose *GUI.MIDDLE* for the vertical alignment and use the *y* coordinate and *height* of the widget as the label's *y* coordinate and *height*. You can get the *y* coordinate and height of a widget using *GUI.GetY* and *GUI.GetHeight*.

Here is an example illustrating the placement of a label at the center of each of four sides of a widget called *w*:

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:200;50")
procedure DoNothing (text : string)
end DoNothing
var w : int := GUI.CreateTextField (50, 15, 100, "", DoNothing)

% These following lines are the important part of the program.
var left := GUI.CreateLabelFull (GUI.GetX (w) - 2, GUI.GetY (w),
    "Left", 0, GUI.GetHeight (w), GUI.RIGHT + GUI.MIDDLE, 0)
var above := GUI.CreateLabelFull (GUI.GetX (w),
    GUI.GetY (w) + GUI.GetHeight (w) + 2, "Above", GUI.GetWidth (w), 0,
    GUI.CENTER + GUI.BOTTOM, 0)
var right := GUI.CreateLabelFull (GUI.GetX (w) + GUI.GetWidth (w) + 2,
    GUI.GetY (w), "Right", 0, GUI.GetHeight (w), GUI.LEFT + GUI.MIDDLE, 0)
var below := GUI.CreateLabelFull (GUI.GetX (w), GUI.GetY (w) - 2,
    "Below", GUI.GetWidth (w), 0, GUI.CENTER + GUI.TOP, 0)
```

Here's the result. Note that the formula for aligning a label with a widget is the same for any type of widget.

Text Box Aligned with Four Labels

Here's an example illustrating aligning a widget with the top of the window. Notice that the label is center aligned with x of 0 and *width* of maxx, and top aligned with a y of maxy and a *height* of 0.

```
var title := GUI.CreateLabelFull (0, maxy, "Title", maxx, 0,  
    GUI.CENTER + GUI.TOP, 0)
```

Finally, here's an example illustrating the placement of a label in the center of the screen. Notice that the label is center aligned with x of 0 and *width* of maxx, and middle aligned with a y of 0 and a *height* of maxy.

```
var title := GUI.CreateLabelFull (0, 0, "Title", maxx, maxy,  
    GUI.CENTER + GUI.MIDDLE, 0)
```

Note that if a label's position or size is changed with *GUI.SetPosition*, *GUI.SetSize* or *GUI.SetPositionAndSize*, it still retains its alignment with respect to its new x , y , *width*, and *height* values.

Canvases

The canvas is a rather unique widget. It is essentially a drawing surface that you place in the window. There are calls using a canvas widget that essentially duplicate all the standard *Draw* module calls, along with calls corresponding to *Font.Draw* and various *Pic* module calls.

The difference is that the calls using the Canvas widget use (0, 0) to mean the bottom left corner of the canvas (not the window) and all drawing is clipped to the canvas (meaning that if you accidentally draw off the canvas, the part of the picture outside the bounds of the canvas will not appear). One of the most common bugs is to accidentally use the actual *Draw* module routines instead of the *GUI.Draw* routines when drawing in a canvas. If the drawing is goes outside the bounds of the Canvas, you have made this mistake.

Another feature of the Canvas widget is that you can specify a procedures to be called whenever a user clicks in the Canvas widget, drags the mouse with the mouse button down and then lets go of the mouse button. These procedures allow your program to respond to mouse activity taking place in the canvas widget.

Here is a program that uses a Canvas to allow the user to draw and a button to allow the user to erase the drawing.

```
% The "Draw" program  
import GUI in "%oot/lib/GUI"
```

```

View.Set ("graphics:300;300")

var oldx, oldy : int
var canvas : int % The drawing canvas.
var clear : int % The clear button.

% Called when the user presses the mouse button in the canvas.
% Sets the initial mouse position.
procedure MouseDown (mx, my : int)
    oldx := mx
    oldy := my
end MouseDown

% Called as the user drags the mouse with the button down in the canvas.
% Draws a line from the previous mouse position to the current position.
procedure MouseDrag (mx, my : int)
    GUI.DrawLine (canvas, oldx, oldy, mx, my, colorfg)
    oldx := mx
    oldy := my
end MouseDrag

% Called when the mouse button is released.
procedure DoNothing (mx, my : int)
end DoNothing

% Called when the clear button is pressed.
procedure Clear
    GUI.DrawCls (canvas)
end Clear

% Create the canvas
canvas := GUI.CreateCanvasFull (10, 30, maxx - 20, maxy - 40, 0,
    MouseDown, MouseDrag, DoNothing)

% Create the clear button
clear := GUI.CreateButton (maxx div 2 - 20, 0, 40, "Clear", Clear)

loop
    exit when GUI.ProcessEvent
end loop

```

Here is the output window after the user has drawn some lines.

Output of Draw.dem

Multiple Windows

Object Oriented Turing for Windows allows for multiple run windows. This can be used to add extra functionality to programs, however there are a few issues that must be understood before multiple windows can be successfully used.

OOT uses the concept of selected windows and active windows. A selected window is determined by the program and is changed by *Window.Select*. The selected window is the window in which all output appears. When a widget is created, it is automatically created in the selected window.

An active window is last window on which the user clicked. The active window is shown by having its title bar highlighted. When a user types, all keystrokes are sent to the active window.

It is entirely possible to have the selected window and the active window be to different windows.

When you call *getch*, *Mouse.ButtonWait*, or any other input routine, OOT checks only the selected window. The GUI Library works around this by selecting all the windows that have widgets in them (one at a time, starting with the active window) and checking each for events.

If you are processing an event from one of several windows, make certain that the correct window is selected before you output your results. Note that the widgets automatically select the correct window, so there is no need to change the selected window before making any calls to the *GUI* module.

The GUI Library Internals

While it is not necessary to know the internals of the GUI Library to use it, we provide this brief overview for those who wish to understand the inner workings of the library.

The GUI Library consists of four parts. The only part visible to the user is the *GUI* module. This is located in "%oot/lib/GUI", where %oot is the directory in which the OOT executable is located. The *GUI* module is essentially a series of procedures that provide a front end to the Widget Module and the Widget Class Library.

The Widget Module is a module called *WidgetModule* that consists of a series of subprograms that cover all the aspects of GUI's that do not pertain to a particular widget. For example, the procedure to change the background color is here, as well as the procedure to process an event. It is located in "%oot/lib/GuiClass/wdgtmod.tu"

The GUI Class Library consists of a series of classes arranged in a hierarchy illustrated in the following figure. Most of the actual Turing code for the GUI Library is located in the Widget Class Library. Each different type of widget has its own class. Widgets that share common behavior have the same parent. For example, both the vertical and horizontal slider have a slider class as a parent. Those classes whose names start with *Generic* are abstract classes. They should not be instantiated themselves. They are used to define common behavior among their subclasses. The Turing source for the classes can be found in the directory "%oot/lib/GuiClass"

The fourth part is the *WidgetGlobals* module. This module that consists mostly of global variables used by the GUI Class Library and the Widget module. It is located in
"%oot/lib/GuiClass/wdgtglob.tu"

Here is an example of how the *GUI* module works: when you create a button using *GUI.CreateButton*, the *CreateButton* function in the *GUI* module creates an object of type *ButtonClass*. (*ButtonClass* is found in the Widget Class Library discussed further down). It then calls the *Initialize* procedure of the *ButtonClass* to initialize the button with the specified parameters. Finally it assigns an ID number to the widget and arranges it in a table for future reference. Here is another example: when you call a procedure like *GUI.Show*, the *Show* procedure takes the widget ID, looks up the object that it represents, and then calls the *Show* procedure of the object.

Students who wish to add new widgets to the GUI library will have to understand the principles of object oriented programming, as they will be adding a new class to the GUI Class Library and then adding new subprograms in the *GUI* module that will call their new classes. (At the very least, a *Create* subprogram will be required for the new widget.)

A suggested project would be to create new versions of the *ButtonClass*, *CheckBoxClass* and *RadioButtonClass* classes that are buttons, check boxes and radio buttons with the new Windows 95/NT appearance. They could be called the *Button95Class*, *CheckBox95Class* and *RadioButton95Class*. Properly written, these new classes should inherit from *ButtonClass*, etc. and contain only those procedures that differ from the base class (presumably the procedures that display the widget).

The GUI Class Library Hierarchy

GUI Module Routines Summary

The routines in the GUI module are divided into several different types. There are the routines to create various widgets, the routines to create menus and menu items, the routines to do general activities (such as processing an event, changing the background color, etc.) and the routines that act on various types of widgets.

Here is the list of the routines that create widgets:

<i>CreateButton</i> , <i>CreateButtonFull</i>	Create a button.
<i>CreateCheckBox</i> , <i>CreateCheckBoxFull</i>	Create a check box.
<i>CreateRadioButton</i> , <i>CreateRadioButtonFull</i>	Create a radio button.
<i>CreatePictureButton</i> , <i>CreatePictureButtonFull</i>	Create a picture button.
<i>CreatePictureRadioButton</i> , <i>CreatePictureRadioButtonFull</i>	Create a picture radio button.
<i>CreateHorizontalSlider</i>	Create a horizontal slider.

<i>CreateVerticalSlider</i>	Create a vertical slider.
<i>CreateHorizontalScrollBar, CreateHorizontalScrollBarFull</i>	Create a horizontal scroll bar.
<i>CreateVerticalScrollBar, CreateVerticalScrollBarFull</i>	Create a vertical scroll bar.
<i>CreateCanvas, CreateCanvasFull</i>	Create a canvas.
<i>CreateTextField, CreateTextFieldFull</i>	Create a text field.
<i>CreateTextBox, CreateTextBoxFull</i>	Create a text box.
<i>CreateLine</i>	Create a line.
<i>CreateFrame</i>	Create a frame.
<i>CreateLabelledFrame</i>	Create a labelled frame.
<i>CreateLabel, CreateLabelFull</i>	Create a label.
<i>CreatePicture</i>	Create a picture.

Here is the list of routines that create menus and menu items:

<i>CreateMenu, CreateMenuFull</i>	Create a menu.
<i>CreateMenuItem, CreateMenuItemFull</i>	Create a menu item.

Here is the list of general routines:

<i>ProcessEvent</i>	Process a single keyboard or mouse down event.
<i>Quit</i>	Tell the program to exit the event loop.
<i>Refresh</i>	Redraw all the widgets on the screen.
<i>SetBackgroundColour</i>	Change the window's background colour.
<i>SetNullEventHandler</i>	Set the null event handler.
<i>SetKeyEventHandler</i>	Set the keystroke event handler.
<i>SetMouseEventHandler</i>	Set the mouse event handler.
<i>HideMenuBar</i>	Hide the menu bar in the window.
<i>ShowMenuBar</i>	Show the menu bar in the window.
<i>GetEventWidgetID</i>	Get the selected widget's ID (used in a widget's action procedure).
<i>GetEventWindow</i>	Get the window that the event took place in (used in a widget's action procedure).
<i>GetEventTime</i>	Get the time (in milliseconds) that the event took place (used in a widget's action procedure).
<i>GetScrollBarWidth</i>	Return the width of a scroll bar.
<i>GetMenuBarHeight</i>	Return the height of a menu bar.
<i>GetVersion</i>	Return the current version number of the GUI module.

Here is a list of routines that act on the widgets and the sort of widgets they act on.

<i>Show</i>	Display the widget.	All
<i>Hide</i>	Hide the widget.	All
<i>GetX</i>	Return the x coordinate of the widget's left edge.	All
<i>GetY</i>	Return the y coordinate of the widget's bottom edge.	All
<i>GetWidth</i>	Return the widget's actual width.	All
<i>GetHeight</i>	Return the widget's actual height.	All
<i>Dispose</i>	Dispose of the widget.	All
<i>SetPosition</i>	Set the widget's position.	All
<i>SetSize</i>	Set the widget's size.	All
<i>SetPositionAndSize</i>	Set the widget's position and size.	All
<i>Enable</i>	Enable the widget to respond to events.	Active Widgets
<i>Disable</i>	Disable the widget from responding to events.	Active Widgets
<i>SetLabel</i>	Set the widget's text label.	Button, Check Box, Radio Button, Label, Labelled Frame
<i>SetDefault</i>	Make the button the default button.	Button
<i>GetCheckBox</i>	Get whether a check box is filled.	Check Box
<i>SetCheckBox</i>	Set a check box to be filled or not.	Check Box
<i>SelectRadio</i>	Select a radio button.	Radio Button, Picture Radio Button
<i>GetSliderValue</i>	Return the current value of the slider.	Slider, Scroll Bar
<i>SetSliderValue</i>	Set the value of the slider.	Slider, Scroll Bar
<i>SetSliderMinMax</i>	Set the slider's minimum and maximum.	Slider, Scroll Bar
<i>SetSliderSize</i>	Set the slider's length (or height).	Slider, Scroll Bar
<i>SetSliderReverse</i>	Reverse the direction of the slider.	Slider, Scroll Bar
<i>SetScrollAmount</i>	Set the scroll bar's thumb size and the scroll amount for arrows/page up and down.	Scroll Bar

<i>DrawArc, DrawBox, DrawCls, DrawDot, DrawFill, DrawFillArc, DrawFillBox, DrawFillMapleLeaf, DrawFillOval, DrawFillPolygon, DrawFillStar, DrawLine, DrawMapleLeaf, DrawOval, DrawPolygon, DrawStar, DrawText</i>	Routines that perform the same function as the <i>Draw</i> module for the Canvas widget.	Canvas
<i>FontDraw</i>	"Font.Draw" for the Canvas widget.	Canvas
<i>PicDraw, PicNew, PicScreenLoad, PicScreenSave</i>	Routines that perform the same function as the <i>Pic</i> module for the Canvas widget.	Canvas
<i>SetXOR</i>	Performs View.Set ("xor") for the Canvas Widget.	Canvas
<i>SetText</i>	Set the text of a text field.	Text Field
<i>SetSelection</i>	Set the selection in the text field.	Text Field
<i>SetActive</i>	Make the text field the active one (where keystrokes will go and where the cursor blinks).	Text Field
<i>ClearText</i>	Clear a text box.	Text Box
<i>AddText</i>	Add text to a text box.	Text Box
<i>AddLine</i>	Add a line of text to a text box.	Text Box

Widgets - Common Routines

All of the procedures in this section can be used with any widget, although some may have no effect (for example `GUI.GetX` on a menu item is meaningless).

GUI.Show Displays a widget. Used in conjunction with *Hide* to hide and show widgets. Hidden widgets cannot get events (i.e. respond to keystrokes or mouse clicks).

GUI.Hide Hides a widget. Used in conjunction with *Show* to hide and show widgets. Hidden widgets cannot get events (i.e. respond to keystrokes or mouse clicks). If an active text field (see text field) is hidden, then any keystrokes in the window will be ignored.

GUI.GetX Returns the x (y) coordinate of the left edge of a widget. Note that this may be different from the x coordinate specified in the widget's *Create* call. For example, if a radio button is created with right justification, the x coordinate in the *Create* method specifies the right edge.

GUI.GetY

Here is a small subprogram that should draw a rectangle entirely around a widget (i.e. no part of the widget should stick out).

```
procedure WidgetRect (widgetID : int)
  const x : int := GUI.GetX (widgetID)
  const y : int := GUI.GetY (widgetID)
  const width : int := GUI.GetWidth (widgetID)
  const height : int := GUI.GetHeight (widgetID)
  Draw.Box (x, y, x + width - 1, y + height - 1, black)
end WidgetRect
```

GUI.GetWidth Returns the actual width (height) of a widget. Note that this may be different from the width specified in the *Create* call (especially since many widgets allow you to specify 0 for the width and let the GUI module determine the necessary width).

GUI.GetHeight

GUI.Dispose Eliminates a widget. It should be called in order to free up any memory that the widget might have allocated. Note that you cannot use the widget after it has been disposed of. If you wish to temporarily get rid of a widget, consider using the *Hide* method and then the *Show* method when you want to use it again.

GUI.SetPosition Moves a widget to the specified location. If the widget is visible, it is moved immediately to the new location. If the widget is hidden, it will appear at the new location when the *Show* procedure is called. Note that the location specified in *GUI.SetPosition* are the same as in the *Create* method. For example, if you had specified a check box to be right justified in the *CreateCheckBoxFull* function, then the location in a call to *SetPosition* would specify the lower-right corner as opposed to the lower-left corner.

- GUI.SetSize** Changes the size of a widget. If the widget is visible, its size is changed immediately, otherwise the widget will appear in its new size when the widget is next made visible. Note that the *width* and *height* parameters are not necessarily the actual width and height of the widget. For example, the *TextField* widget ignores the *height* parameter, calculating the widget's actual height from the height of the text in the *TextField*.
- GUI.SetPositionAndSize** Changes the position and size of the widget simultaneously. It works the same way as the *SetPosition* and *SetSize* procedures.

Widgets - Buttons

Output of Buttons.dem program

The button widget is used to implement a textual button. When you click on a button, the button's *action procedure* is called. If a button is given a short cut, then entering the keystroke will cause the *action procedure* to be called. It will not visibly cause the button to depress.

If a button's width or height is set to zero (or not specified at all), then the button is shaped to fit the text.

A button can be the default button for a window. If that is the case, then the button will be drawn with a thicker border around it and if the user presses ENTER, then the button's *action procedure* will be called.

When a button is not enabled, the text in the button is grayed out and the button no longer responds to any mouse clicks or keystrokes until the button is enabled again.

GUI.CreateButton Creates and displays a button. *GUI.CreateButton* specifies the location, width, text and action procedure of the button.

GUI.CreateButtonFull Creates and displays a button. *GUI.CreateButtonFull* also specifies the height, keyboard shortcut and default status of the button.

GUI.Enable Enables (disables) a button. Disabled buttons have their text grayed out and cannot get events (i.e. respond to keystrokes or mouse clicks).

GUI.SetLabel Changes the text of a button.

GUI.SetDefault Sets the "default status" of a button. If a button is the default button, then it is drawn with a heavy outline and it is activated when the user presses ENTER (RETURN on a Macintosh).

Widgets - Check Boxes

Output of ChkBoxes.dem

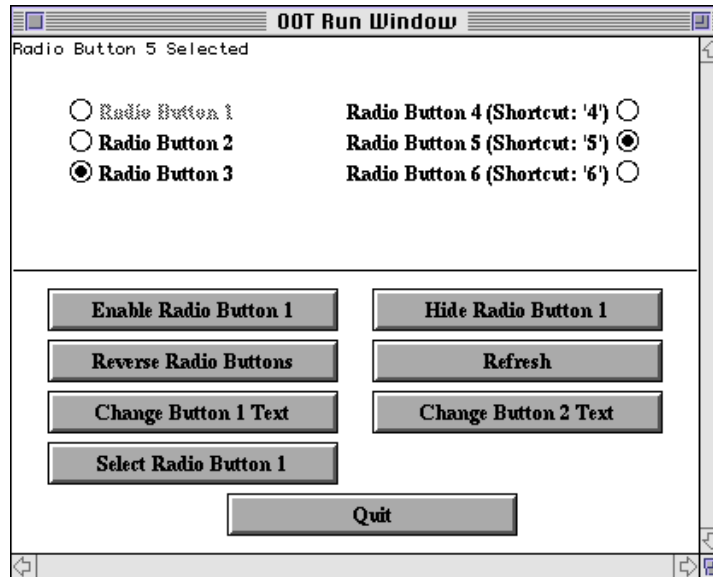
The check box widget is used to implement a check box that can be set or unset. When you click on a check box, the status of the check box flips from set to unset and back again and the check box's *action procedure* is called with the new status as a parameter. If a check box is given a short cut, then entering the keystroke will cause the check box to change status and the *action procedure* to be called. The new status will be displayed immediately.

A check box's size is not specified during creation. It is determined based on the size of the text. Instead the user specifies the lower-left corner of the check box (or the lower-right if the check box is right justified).

When a check box is not enabled, the label beside the check box is grayed out and the check box no longer responds to any mouse clicks or keystrokes until the check box is enabled again.

- GUI.CreateCheckBox** Creates and displays a left aligned (check box to the left of the label) check box. *GUI.CreateCheckBox* specifies the location, text and action procedure of the check box.
- GUI.CreateCheckBoxFull** Creates and displays a check box. *GUI.CreateCheckBoxFull* also specifies the alignment of the check box (whether the checkbox is the right or left of the text) and the check box's keyboard shortcut.
- GUI.Enable** Enables (disables) a check box. Disabled check boxes have their text grayed out and cannot get events (i.e. respond to keystrokes or mouse clicks).
- GUI.Disable**
- GUI.SetLabel** Changes the text of a check box.
- GUI.GetCheckBox** Returns a check box's status. If the check box is set (has an X in it), *GUI.GetCheckBox* returns true, otherwise it returns false.
- GUI.SetCheckBox** Sets the status of a check box. It calls the check box's *action procedure* with the new status and redraws the widget with the new status.

Widgets - Radio Buttons



Output from Radios.dem

The radio button widget is used to implement a set of buttons of which one and only one button must be selected at all times. (Think old-style radio station button. Selecting one "deselects" the previously-selected station.) When you click on a radio button, any other radio button that is part of the set is deselected and the radio button's *action procedure* is called. If a radio button is given a short cut, then entering the keystroke will cause the radio button to be selected (and any other radio button in the group to be de-selected) and the *action procedure* to be called. The newly-selected or deselected radio buttons will be displayed immediately.

When a radio button is created, the widget ID of another radio button must be supplied. A value of zero for the widget ID indicates that this radio button is part of a new group. The widget ID must be the ID of the last radio button added to the group. Because radio buttons are almost always placed in groups you can specify -1 for the x and y coordinates and the radio button will be placed just below the previous radio button and retain the same alignment. When a group of radio buttons is selected, the first radio button created in the group will be the selected one. You can change this by using the *GUI.SelectRadio* procedure to select a different one.

A radio button's size is not specified during creation. It is determined based on the size of the text. The user specifies the lower-left corner of the radio button (or the lower-right if the radio button is right justified).

When a radio button is not enabled, the label beside the radio button is grayed out and the radio button no longer responds to any mouse clicks or keystrokes until the radio button is enabled again.

GUI.CreateRadioButton	Creates and displays a left aligned (circle to the left of the label) radio button. <i>GUI.CreateRadioButton</i> specifies the location, text, the radio button to be joined to and the action procedure of the radio button.
GUI.CreateRadioButtonFull	Creates and displays a radio button. <i>GUI.CreateRadioButtonFull</i> also specifies the alignment of the radio button (whether the circle is the right or left of the text) and the radio button's keyboard shortcut.
GUI.Enable GUI.Disable	Enables (disables) a radio button. Disabled radio buttons have their text grayed out and cannot get events (i.e. respond to keystrokes or mouse clicks).
GUI.SetLabel	Changes the text of a radio button.
GUI.SelectRadio	Selects a radio button. The previously-selected radio button is "de-selected". The <i>action procedure</i> of the radio button is called.

Widgets - Picture Buttons

Picture button widgets

The picture button widget (hereafter simply called a button) is simply a button with a picture on it instead of text. The picture must be created by the program beforehand using *Pic.New* or *Pic.FileNew*. The resulting picture can then be used as a parameter to *GUI.CreatePictureButton*. In general, pictures should be a maximum of about 30 pixels high and wide, although there is no built in limit in the GUI library.

When you click on a picture button, the picture button's *action procedure* is called. If a picture button is given a short cut, then entering the keystroke will cause the *action procedure* to be called. It will not visibly cause the button to depress.

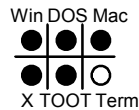
If a button's width or height is set to zero (or not specified at all), then the button is shaped to fit the picture.

When a picture button is not enabled, the picture button is grayed out and the picture button no longer responds to any mouse clicks or keystrokes until the button is enabled again.

GUI.CreatePictureButton	Creates and displays a picture button. The button is automatically sized to fit the picture. If you need to know the precise size of the button, use the <i>GUI.GetWidth</i> and <i>GUI.GetHeight</i> functions. <i>GUI.CreatePictureButton</i> specifies the location, picture id and action procedure of the button.
GUI.CreatePictureButtonFull	Creates and displays a picture button. <i>GUI.CreatePictureButtonFull</i> also specifies the width, height and

	keyboard shortcut of the button. It also specifies whether the button picture should be merged with the background color or not.
GUI.Enable	Enables (disables) a picture button. Disabled picture buttons are
GUI.Disable	grayed out and cannot get events (i.e. respond to keystrokes or mouse clicks).

Widgets - Picture Radio Buttons



Picture radio button widgets

The picture radio button widget (hereafter simply called a button) is simply a picture button (see **Widget - Picture Button**) that has the behavior of a radio button. This means that one and only one picture radio button of a group is selected at any time. A selected picture radio button is displayed as being pressed.

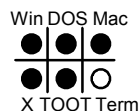
When you click on a picture button, the previously-selected picture radio button will be de-selected and the new picture button's *action procedure* is called. If a picture button is given a short cut, then entering the keystroke will cause the *action procedure* to be called and the picture radio button will be drawn selected.

GUI.CreatePictureRadioButton Creates and displays a picture radio button. The button is automatically sized to fit the picture. If you need to know the precise size of the button, use the *GUI.GetWidth* and *GUI.GetHeight* functions. *GUI.CreatePictureRadioButton* specifies the location, picture id and action procedure of the button as well as the radio picture button to be joined to.

GUI.CreatePictureRadioButtonFull Creates and displays a picture radio button. *GUI.CreatePictureRadioButtonFull* also specifies the width, height and keyboard shortcut of the button. It also specifies whether the button picture should be merged with the background color or not.

GUI.Enable	Enables (disables) a picture radio button. Disabled picture buttons are
GUI.Disable	grayed out and cannot get events (i.e. respond to keystrokes or mouse clicks).

Widgets - Sliders



Output of Sliders.dem

Sliders are the equivalent of a volume control on a stereo. To control a slider, the user simply clicks on the control knob and slides the control left and right (up and down for a vertical slider). Whenever the user slides the control knob, the *action procedure* of the widget is called with the current value as a parameter.

The range of values that the slider will give is determined by the *min* and *max* parameters in the *Create* call. The left side of the slider (bottom for vertical sliders) represents the minimum value, while the right (top) represents the maximum value.

In some instances, you will want the reverse to be true (right/top is minimum). In that case, call the *GUI.SetSliderReverse* procedure to flip the values of the slider.

Sliders always have a fixed height (for horizontal sliders) or width (for vertical sliders). The length parameter in the *Create* call specifies how long the slider should be.

GUI.CreateHorizontalSlider Creates and displays a horizontal (left-right) slider.
GUI.CreateHorizontalSlider specifies the location, length, minimum and maximum values for the slider, the initial value of the slider, and the action procedure of the slider.

GUI.CreateVerticalSlider Creates and displays a vertical (up-down) slider.
GUI.CreateVerticalSlider specifies the location, length, minimum and maximum values for the slider, the initial value of the slider, and the action procedure of the slider.

GUI.Enable Enables (disables) a slider. Disabled sliders cannot get events
GUI.Disable (i.e. respond to mouse clicks).

GUI.GetSliderValue Returns the current value of a slider.

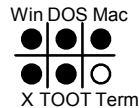
GUI.SetSliderValue Sets the value of a slider. It moves the control knob on the slider to the appropriate location and calls the slider's *action procedure* with the new value.

GUI.SetSliderMinMax Sets the minimum and maximum values of a slider. It redraws the control knob to take into account the new minimum and maximum. If the current value of the slider is outside the new min/max, then the value is adjusted appropriately.

GUI.SetSliderSize Changes the length of a slider. Redraws the slider and changes the position of the control knob to take into account the new size of the slider.

GUI.SetSliderReverse Sets a slider into (or out of, if already into) "reverse mode". Normally, a slider is at its minimum value when the control knob is on the left side (bottom for a vertical slider). This reverses it, so the minimum value is when the slider is at the right side (top for vertical sliders) of the track. Calling this routine a second time reverses it back to normal. This procedure redraws the slider to move the control knob to its new location.

Widgets - Scroll Bars



Output of ScrlBrs.dem

Scroll bars are usually used to allow a user to see a piece of a document that cannot be displayed on the screen in its entirety. The picture above shows the scroll bars appearance. To control a scroll bar, there are a few choices: the user can click on the thumb (the box in the scroll bar) and slide it up and down, or the user can click in the scroll bar itself above or below the thumb (in which case the thumb is moved up or down one "page"), or the user can click on the up or down arrows at the ends of the scroll bar (in which case the thumb is moved up one "line"). The programmer defines a page or a line. When the value of the scroll bar changes, the *action procedure* of the scroll bar is called with the new value as a parameter.

The range of values that the scroll bar will give is determined by the *min* and *max* parameters in the *Create* call. The left side of the scroll bar (bottom for vertical scroll bars) represents the minimum value, while the right (top) represents the maximum value. There is also the "thumb size". This represents the range of values that can be seen at once on the screen.

For example, if you have a window that can display 20 lines of text at once and there are 100 lines of text, you would set *min* to 1, *max* to 100, and *thumbSize* to 20. The value returned by the scroll bar would then be the line number of the first line on the screen to be displayed. When the scroll bar was at its maximum value, it would return 81, since by doing so, lines 81-100 would be displayed.

When a scroll bar is disabled or the scroll bar's thumb size is greater than the difference between the minimum and maximum values (i.e. the item being scrolled fits in the window), the scroll bar is deactivated. The bar is drawn in white rather than gray and the arrows are grayed out. The scroll bar does not respond to mouse clicks.

In some instances, you will want the minimum and maximum values of the scroll bar to be reversed (right/top is minimum). In that case, call the *GUI.SetSliderReverse* procedure to flip the values of the scroll bar.

Scroll bars always have a fixed height (for horizontal scroll bars) or width (for vertical scroll bars). To get the scroll bar's width, use the *GUI.GetScrollBarWidth* function. The length parameter in the *Create* call specifies how long the scroll bar should be.

GUI.CreateHorizontalScrollBar Creates and displays a horizontal (left-right) scroll bar.

GUI.CreateHorizontalScrollBar specifies the location, length, minimum and maximum values for the scroll bar, the initial value of the scroll bar, and the scroll bar's action procedure.

By default, the arrow increment (the amount the value is changed when the scrolling arrows are pressed) is set to one. The page up/down increment (the amount the value is changed when the user

clicks in the bar to the right or left of the thumb) is set to one quarter the difference between the minimum and the maximum. The "thumb size" is set to zero (see the description of scroll bars for an explanation of the thumb size).

GUI.CreateVerticalScrollBar Creates and displays a vertical (up-down) scroll bar. *GUI.CreateVerticalScrollBar* specifies the location, length, minimum and maximum values for the scroll bar, the initial value of the scroll bar, and the scroll bar's action procedure.

By default, the arrow increment (the amount the value is changed when the scrolling arrows are pressed) is set to one. The page up/down increment (the amount the value is changed when the user clicks in the bar to the right or left of the thumb) is set to one quarter the difference between the minimum and the maximum. The "thumb size" is set to zero (see the description of scroll bars for an explanation of the thumb size).

GUI.CreateHorizontalScrollBarFull Creates and displays a horizontal (left-right) scroll bar. *GUI.CreateHorizontalScrollBarFull* also specifies the arrow increment, page increment, and thumb size for the scroll bar.

GUI.CreateVerticalScrollBarFull Creates and displays a horizontal (left-right) scroll bar. *GUI.CreateVerticalScrollBarFull* also specifies the arrow increment, page increment and thumb size for the scroll bar.

GUI.Enable Enables (disables) a scroll bar. Disabled scroll bars cannot get events
GUI.Disable (i.e. respond to mouse clicks).

GUI.GetSliderValue Returns the current value of a scroll bar.

GUI.SetSliderValue Sets the value of a scroll bar. It moves the control knob on the scroll bar to the appropriate location and calls the scroll bar's *action procedure* with the new value.

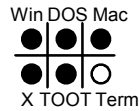
GUI.SetSliderMinMax Sets the minimum and maximum values of a scroll bar. It redraws the control knob to take into account the new minimum and maximum. If the current value of the scroll bar is outside the new min/max, then the value is adjusted appropriately.

GUI.SetSliderSize Changes the length of a scroll bar. Redraws the scroll bar and changes the position of the control knob to take into account the new size of the scroll bar.

GUI.SetSliderReverse Sets a scroll bar into (or out of, if already into) "reverse mode". Normally, a scroll bar is at its minimum value when the control knob is on the left side (bottom for a vertical scroll bar). This reverses it, so the minimum value is when the scroll bar is at the right side (top for vertical scroll bars) of the track. Calling this routine a second time reverses it back to normal. This procedure redraws the scroll bar to move the control knob to its new location.

GUI.SetScrollBarAmount Sets a scroll bar's arrow increment, page increment, and thumb size. Redraws the scroll bar to take into account the new thumb size.

Widgets - Canvases



Output of Canvases.dem

A canvas is a drawing surface for use by the program. It differs from just using the window surface to draw on in that (0, 0) represents the lower-left corner of the canvas and all drawing is clipped to the canvas. (This means that if you accidentally attempt to draw outside of the canvas, it will not actually draw beyond the border of the canvas.)

Canvases have procedures that emulate all the procedures in the Draw module as well as a procedure to emulate Font.Draw, Pic.Draw, Pic.New, Pic.ScreenLoad, and Pic.ScreenSave.

You can get mouse feedback from a canvas. Using the *GUI.CreateCanvasFull* method, you can specify three routines that are called when the mouse button is depressed while pointing in a canvas. One routine will be called when the user presses the mouse button down in a canvas. Another routine will be called while the user drags the mouse with the mouse button down. This routine is repeatedly called whenever the mouse changes position while the mouse button is down. The last routine is called when the mouse button is released. All three routines take an *x* and *y* parameter, which is the location of the mouse with respect to the canvas (i.e. (0, 0) is the lower-left corner of the canvas).

GUI.CreateCanvas Creates and displays a canvas. *GUI.CreateCanvas* specifies the location and size of the canvas. The canvas will have a line border around it.

GUI.CreateCanvasFull Creates and displays a canvas. *GUI.CreateCanvasFull* also specifies the type of border and three procedures to be called when a mouse is pressed, dragged or released on the canvas.

GUI.Enable Enables (disables) a canvas. Disabled canvases cannot get events (i.e. respond to mouse clicks). If no mouse routines were specified (i.e. the canvas was created with *GUI.CreateCanvas* and not *GUI.CreateCanvasFull*) this routine essentially does nothing.

GUI.Disable

GUI.DrawArc	All these routines draw to a canvas in the same manner as the similarly named Draw..., Pic... and Font.Draw subprograms.
GUI.DrawBox	
GUI.DrawCIs	All coordinates are based on the canvas and all drawing is clipped to the canvas drawing surface. If the canvas is in "xor mode", all the drawing will be done with "xor" set. (See <i>View.Set</i> for more information about "xor".)
GUI.DrawDot	
GUI.DrawFill	
GUI.DrawFillArc	
GUI.DrawFillBox	
GUI.DrawFillMapleLeaf	
GUI.DrawFillOval	
GUI.DrawFillPolygon	
GUI.DrawFillStar	
GUI.DrawLine	
GUI.DrawBox	
GUI.DrawMapleLeaf	
GUI.DrawOval	
GUI.DrawPolygon	
GUI.DrawStar	
GUI.DrawText	
GUI.FontDraw	
GUI.PicDraw	
GUI.PicNew	
GUI.PicScreenLoad	
GUI.PicScreenSave	
GUI.SetXOR	Sets the "xor mode" of a canvas. When in "xor mode", all the <i>Draw...</i> procedures of a canvas are treated as if the <i>View.Set</i> ("xor") statement had been executed before the <i>Draw</i> procedure.

Widgets - Text Fields



Output of TextFlds.dem

A text field is a box for entering one line of text. When the user presses ENTER, the text field's *action procedure* is called.

Only one text field is active at a time. The active text field has a blinking cursor (or its selection highlighted). If a keystroke occurs when a window has an active text field in it, the keystroke will be directed to the active text field. You can change which text field is active with the *GUI.SetActive* procedure or by simply clicking on another text field with the mouse.

When multiple text fields are created in a window, the first text field created is active when the program begins.

The current version of the text field does not support cut and paste or keyboard commands to extend the selection.

Because strings are a maximum of 255 characters, this is the maximum number of characters in a text field.

The TAB character cycles between different text fields in a window. It cycles through the text fields in the order in which they were created. BACK TAB (shift+TAB) cycles through the fields in reverse order.

GUI.CreateTextField Creates and displays a text field. *GUI.CreateTextField* specifies the location, width, initial text string, and action procedure of the text field. The height of the text field is determined by the height of the font used by the text field. The text field will have a line border around it.

GUI.CreateTextFieldFull Creates and displays a text field. *GUI.CreateTextFieldFull* also specifies the type of border, font for entered text, and kind of input restriction (integer only, etc.)

GUI.Enable Enables (disables) a text field. Disabled picture buttons are grayed out and cannot get events (i.e. respond to keystrokes or mouse clicks).

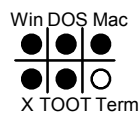
GUI.SetText Sets the text of a text field. The cursor is set the beginning of the text.

GUI.GetText Returns the current text of a text field.

GUI.SetSelection Sets the selection (the selected text) in a text field.

GUI.SetActive Makes a text field active.

Widgets - Text Boxes

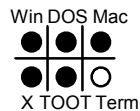


Output of TextBxs.dem

A text box is a box used for displaying larger quantities of text. It has both vertical and horizontal scroll bars to allow the user to scroll through all the text in the box.

GUI.CreateTextBox	Creates and displays a text box. <i>GUI.CreateTextBox</i> specifies the location and size of the text box. The text box will have a line border around it.
GUI.CreateTextBoxFull	Creates and displays a text box. <i>GUI.CreateTextBoxFull</i> also specifies the type of border and the font for displayed text.
GUI.ClearText	Clears all the text in a text box.
GUI.AddText	Adds text to the current line of the text box. Does not add a newline after the text. Equivalent to put text This scrolls the text box (if necessary) so that the added text is now visible. To move the cursor to the end of the text without adding any extra text, call <i>GUI.AddText</i> with "" for the <i>text</i> parameter.
GUI.AddLine	Adds text to the current line of the text box followed by a newline. Equivalent to put text . This scrolls the text box (if necessary) so that the added text is now visible.

Widgets - Lines

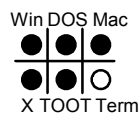


Output of Lines.dem

Lines are organizational elements that make the window look better and help organize the GUI elements.

GUI.CreateLine	Creates and displays a line. <i>GUI.CreateLine</i> specifies the end points of the line (which must be either vertical or horizontal) and the type of line.
-----------------------	---

Widgets - Frames

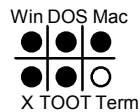


Output from Frames.dem

Frames are organizational elements that make the window look better and help organize the GUI elements. Frames and labelled frames are the only widgets in which other widgets can be placed.

GUI.CreateFrame Creates and displays a frame. *GUI.CreateFrame* specifies the coordinates of the lower-left and upper-right corner of the frame and the type of border of the frame.

Widgets - Labelled Frames



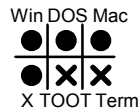
Output from LblFrms.dem

Labelled frames are organizational elements that make the window look better and help organize the GUI elements. Frames and labelled frames are the only widgets in which other widgets can be placed.

GUI.CreateLabelledFrame Creates and displays a labelled frame. *GUI.CreateLabelledFrame* specifies the coordinates of the lower-left and upper-right corner of the frame, the type of border of the frame, and the text of the frame's label.

GUI.SetLabel Changes the text of a labelled frame.

Widgets - Labels



Output from Labels.dem

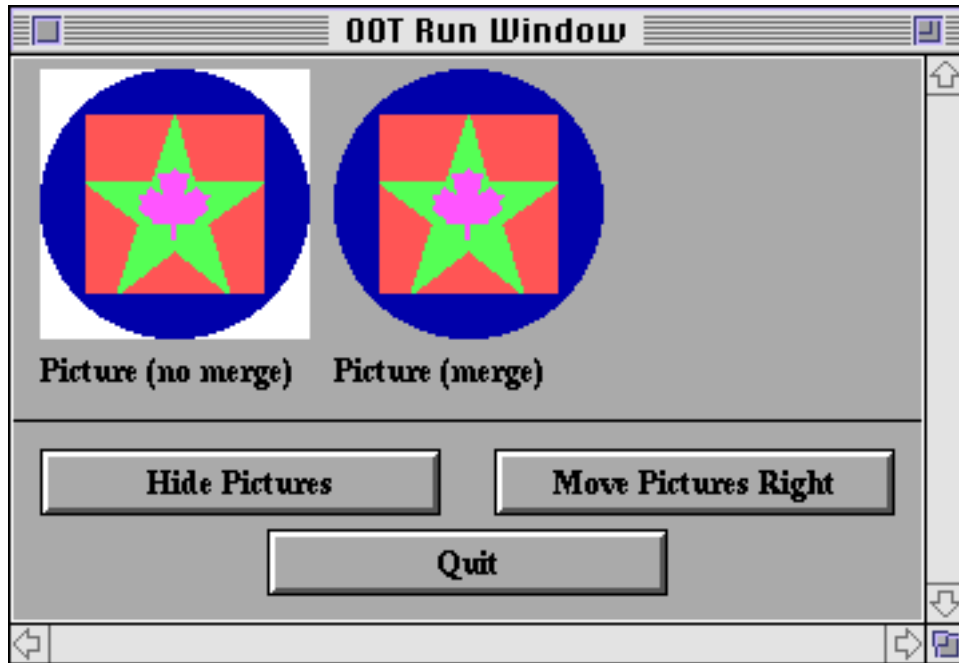
Labels are organizational elements that make the window look better and help organize the GUI elements. They are simply text placed in a window. To aid in aligning text with various widgets, it is possible to align text in a larger region (as shown in the figure).

GUI.CreateLabel Creates and displays a label. *GUI.CreateLabel* specifies the lower-left corner of the text and the text itself. The system font is used to display the label.

GUI.CreateLabelFull Creates and displays a label. *GUI.CreateLabelFull* also specifies the width, height, alignment, and font for the label. The width and height are specified for alignment purposes.

GUI.SetLabel Changes the text of a label.

Widgets - Pictures

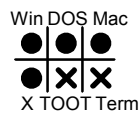


Output from Pictures.dem

Pictures are organizational elements that make the window look better and help organize the GUI elements. They are simply a picture placed in a window. The pictures are specified using a picture ID from any of the **Pic...** subprograms.

GUI.CreatePicture Creates and displays a picture. *GUI.CreatePicture* specifies the location, picture ID, and whether the picture should be merged with the background.

Widgets - Menus



Output from Menus.dem

Menus are used in most modern interfaces. In order to create a full set of menus, you must create the menu and then create the menu items in that menu. The menus are automatically added to the menu bar of the selected menu.

Menu items are the individual entries of a menu. To create menus for a window, you must create a menu, then create the menu items for that menu, then create the next menu, then the items for that menu, etc. All menu items are automatically added to the last menu and after the last menu item of the currently selected (not active!) window.

When you select an item in a menu, the *action procedure* of the item is called. The *action procedure* has no parameters.

As of the v1.0 release of the GUI Library, it is an error to create a menu item without having created a menu first. In future releases it will be possible to create menus and attach and remove them from menu bars when desired.

Menus and menu items can be enabled and disabled. A disabled menu item is grayed out. When the user selects the menu, all items in the menu appear disabled and cannot be selected. A disabled menu item is grayed out when the menu is displayed. The user cannot select the menu item.

Separators in a menu appear as a solid line across the menu. These are created by creating a menu item whose text is three dashes "---".

GUI.CreateMenu Creates and displays a menu. The menu will be added after the other menus in the menu bar. If there are no previous menus, then a menu bar is automatically created and the menu added. *GUI.CreateMenu* specifies the text that will appear in the menu bar. It is suggested that the text not have any spaces in it.

GUI.Enable Enables (disables) a menu. Disabled menus are grayed out in the menu bar. If selected, all the menu items in the menu bar appear disabled and cannot be selected.

GUI.CreateMenuItem Creates a menu item. *GUI.CreateMenuItem* specifies the text of the menu item and the *action procedure* to be called when the menu item is selected. The menu item will be added to the last menu after the other menu items in the menu. If there are no menus defined, an error results.

GUI.CreateMenuItemFull Creates a menu item. *GUI.CreateMenuItemFull* also specifies a shortcut keystroke.

GUI.Enable Enables (disables) a menu item. Disabled menu items are grayed out when the menu is displayed and cannot be selected by the user.

Widgets - General Routines

The following procedures are included in the GUI module but do not relate to a specific widget.

GUI.ProcessEvent This function processes a single event (a mouse button press or a keystroke). If the event activates a widget, then the *action procedure* of the widget is called. To find out which widget was activated and called the *action procedure* (necessary if several widgets have the same *action*

procedure), you can call *GUI.GetEventWidgetID*. To get the exact time that the event occurred, you can call *GUI.GetEventTime*. To get the window in which the event took place, you can call *GUI.GetEventWindow*.

If a mouse click occurred, but did not activate any widget, then the default mouse event handler is called. By default, this does nothing. However, if you want your program to respond to mouse events that do not affect a widget, call *GUI.SetMouseEventHandler* to specify your own default mouse event handler.

If a keystroke occurred, but did not activate any widget (i.e. it wasn't a short cut for a widget and there are no text fields in the window) then the default keystroke handler is called. By default, this does nothing. However, if you want your program to respond to keystroke events that do not affect a widget, call *GUI.SetKeyEventHandler* to specify your own default key event handler.

If no event occurred, then the null event handler is called. By default, this does nothing. However, if you want your program to perform some action repetitively when it is not doing anything else, then call *GUI.SetNullEventHandler* to specify your own null event handler. The null event handler is often used for such things as updating a clock and making certain that music is playing in the background.

GUI.Quit This procedure causes *GUI.ProcessEvent* to return **true**. If the program is structured properly with a

```

loop
    exit when GUI.ProcessEvent
end loop

```

at the end of the program, then the program will exit the loop after finishing the current *action procedure*. This procedure is usually called from the *action procedure* of a Quit button or Exit menu item.

GUI.Refresh This routine redraws all the widgets in the currently-selected window. This is used when some form of drawing may have overwritten the widgets in a window. It is used by the GUI Library to redraw all the widgets when the background color of a window has changed.

GUI.SetBackgroundColor Changes the background color of the currently-selected window. (Both spellings of color are acceptable.) This does *not* change the value of color 0 in the window. Instead it fills the entire window with the new background color and then redraws all the widgets. The usual background color outside of white is *gray*.

GUI.SetNullEventHandler Sets the new null event handler. The specified procedure will be called every time *GUI.ProcessEvent* is called and there is no keystroke or mouse button pressed.

GUI.SetMouseEventHandler Sets the new default mouse event handler. The specified procedure will be called every time *GUI.ProcessEvent* is called and there is a mouse button pressed which is not handled by any widget.

- GUI.SetKeyEventHandler** Sets the new default keystroke event handler. The specified procedure will be called every time *GUI.ProcessEvent* is called and there is a keystroke which is not handled by any widget.
- GUI.HideMenuBar** Hides the menu bar in the selected window. No menu items can be selected when the menu bar is hidden. (Menu item shortcuts will be ignored when the menu bar is hidden.)
- GUI.ShowMenuBar** Shows the menu bar in the selected window.
- GUI.GetEventWidgetID** Returns the widget ID of the widget that was activated by the mouse button press or the keystroke. This function should only be called in an *action procedure*, as it will return -1 when there is no event that activated a widget being processed.
- GUI.GetEventWindow** Returns the window ID of the window in which the event (mouse button or keystroke) took place. This function should only be called in an *action procedure* or in a default mouse or keystroke event handler, as it will return -1 when there is no event being processed.
- GUI.GetEventTime** Returns the time in milliseconds when the event (mouse button or keystroke) took place. This value is the same value that *Time.Elapsed* returns if called when the event was processed. This function should only be called in an *action procedure* or in a default mouse or keystroke or null event handler, as it will return -1 when there is no event being processed.
- GUI.DisplayWhenCreated** Sets whether widgets are automatically displayed when created, or whether *GUI.Show* must be called first. By default, this is set to true (widgets are displayed when created). However, there may be times when you want to create a widget and then make several additional calls before displaying the widget.
- GUI.CloseWindow** Closes a window with widgets in it. This procedure automatically disposes of any widgets in the window and makes certain that the GUI Library recognizes that the window no longer exists. This procedure will call *Window.Close*, so there is no need for the user to do so.
- GUI.GetScrollBarWidth** Returns the width of a scroll bar. Useful when placing a scroll bar widget beneath another object.
- GUI.GetMenuBarHeight** Returns the height of the menu bar. Useful when placing widgets to make certain that they do not overlap the menu.
- GUI.GetVersion** Returns the current version of the GUI module. Because the GUI module is expected to grow, new versions will probably be made available at Holt Software's web site located at <http://www.holtsoft.com/turing>. If you wish to use features that do not appear in earlier versions of the library, you can have your program check that the current available version meets the programs needs. *GUI.GetVersion* returns an integer from 100 - 999 and is read as 1.00 to 9.99.

Chapter 6

Importing Graphics into a Turing Program

Introduction

Graphics created outside of Turing can be imported into Turing programs. Typically, these graphics are created in programs like CorelDraw or Microsoft Paint or downloaded from the internet.

To import graphics into WinOOT (Object Oriented Turing running under Microsoft Windows), the graphics must be in BMP file format. BMP is a common MS Windows file format suitable for images including clipart and digitized photographs. Almost all conversion programs can convert images to the BMP format.

To import graphics into DOS OOT (Object Oriented Turing running under MS-DOS), the graphics must be in PCX file format. PCX is a common file format suitable for images including clipart and digitized photographs. Almost all conversion programs can convert images to the PCX format.

To import graphics in MacOOT (Object Oriented Turing running on the Macintosh), the graphics must be in PICT format, which is the standard picture format for the Macintosh. Almost all Macintosh conversion programs can convert images to PICT format.

Occasionally, pictures will be obtained in a file format that is different from the one required by Turing. In such a case, a picture format conversion utility is required. There are a wide variety of freeware and shareware picture format conversion utilities available. Checking the internet is often the best way to find such programs. The site www.tucows.com provides large numbers of such programs.

Once the image has been converted into BMP, PCX, or PICT format (depending on whether a PC or a Macintosh is being used), then the file can be imported into Turing.

Importing BMP Images in WinOOT

WinOOT displays (and saves) BMP images using the **Pic.ScreenLoad** and **Pic.ScreenSave** procedures located in the **Pic** module.

The BMP format is the standard graphics interchange format for Microsoft Windows. Almost any graphics program created for MS Windows supports it in one form or another.

Pic.ScreenLoad

The **Pic.ScreenLoad** procedure has the following format:

Pic.ScreenLoad (*filename* : **string**, *x*, *y*, *mode* : **int**)

This procedure is the equivalent of **drawpic**, except that it draws from a file instead of an array. It will display the BMP file, with the lower-left corner of the picture being (*x*, *y*). For full screen pictures, *x* and *y* should be 0. The *mode* parameter must one of *picCopy*, in which case the image displays over top of whatever is on the screen, *picMerge*, which functions like *picCopy* except that any occurrence of the background color in the picture is not drawn to the screen, or *picXor*, in which case the image is "xor"-ed with the underlying image on the screen.

If the procedure is unable to open or display the file it will not display an image. Instead **Error.Last** will be set to non-zero. It is a good idea to always check **Error.Last** to make certain that it is non-zero.

Here is a small example that displays the file "FOREST.BMP" at location (50, 50).

```
Pic.ScreenLoad ("FOREST.BMP", 50, 50, picCopy)
if Error.Last not= 0 then
    put "Error displaying image: ", Error.LastMsg
end if
```

Pic.ScreenLoad requires that either the file name end with ".BMP" or the file name be prefaced with "BMP:". For example, if the BMP file was called "FOREST.PCT", then the **Pic.ScreenLoad** line would look like:

```
Pic.ScreenLoad ("BMP:FOREST.PCT", 50, 50, picCopy)
```

In the current implementation of WinOOT (WinOOT 3.0), Turing remaps the palette of the image to the palette of the window on the screen. By default, WinOOT run windows have a palette of only 16 colors. This means that any BMP image drawn to the screen will be mapped to the base 16 colors, which may render the image less pleasing than desired. To add more colors to the palette, and thus increase the fidelity of the image, it is necessary to use **RGB.AddColor**. If you know the palette of the BMP image, you can add the specific colors desired. If you want to add a wider general palette to the screen, you can use the following:

```
var dummy : int
for red : 0 .. 5
    for green : 0 .. 5
        for blue : 0 .. 5
            dummy := RGB.AddColor (red / 5.0, green / 5.0, blue / 5.0)
        end for
    end for
end for
```


Pic.ScreenSave

The **Pic.ScreenSave** procedure has the following format:

```
Pic.ScreenSave (x1, y1, x2, y2 : int, filename : string)
```

This procedure is the equivalent of **takepic**, except it copies the screen to a file instead of an array. The section of the screen located in the rectangle specified by (*x1*, *y1*) - (*x2*, *y2*) is copied to a BMP file located at *filename*. As with **Pic.ScreenLoad**, the file name must end with ".BMP" or be prefaced with "BMP:".

Here is a small example that writes the entire screen to a file called "BACKGRND.BMP".

```
Pic.ScreenSave (0, 0, maxx, maxy, "BACKGRND.BMP")
```

Importing PICT Images in MacOOT

MacOOT displays (and saves) PICT images using the **Pic.ScreenLoad** and **Pic.ScreenSave** procedures located in the **Pic** module.

The PICT format is the standard graphics interchange format on the Macintosh. Almost any graphics program created for the Macintosh supports it in one form or another.

Pic.ScreenLoad

The **Pic.ScreenLoad** procedure has the following format:

```
Pic.ScreenLoad (filename : string, x, y, mode : int)
```

This procedure is the equivalent of **drawpic**, except that it draws from a file instead of an array. It will display the PICT file, with the lower-left corner of the picture being (*x*, *y*). For full screen pictures, *x* and *y* should be 0. The *mode* parameter must be either *picCopy*, in which case the image displays over top of whatever is on the screen, or *picXor*, in which case the image is "xor"-ed with the underlying image on the screen.

If the procedure is unable to open or display the file it will not display an image. Instead **Error.Last** will be set to non-zero. It is a good idea to always check **Error.Last** to make certain that it is non-zero.

Here is a small example that displays the file "FOREST.PICT" at location (50, 50).

```
Pic.ScreenLoad ("FOREST.PICT", 50, 50, picCopy)  
if Error.Last not= 0 then  
    put "Error displaying image: ", Error.LastMsg  
end if
```

Pic.ScreenLoad requires that either the file name end with ".PICT" or the file name be prefaced with "PICT:". For example, if the PICT file was called "FOREST.PCT", then the **Pic.ScreenLoad** line would look like:

Pic.ScreenLoad ("PICT:FOREST.PCT", 50, 50, *picCopy*)

Pic.ScreenSave

The **Pic.ScreenSave** procedure has the following format:

Pic.ScreenSave (*x1*, *y1*, *x2*, *y2* : **int**, *filename* : **string**)

This procedure is the equivalent of **takepic**, except it copies the screen to a file instead of an array. The section of the screen located in the rectangle specified by (*x1*, *y1*) - (*x2*, *y2*) is copied to a PICT file located at *filename*. As with **Pic.ScreenLoad**, the file name must end with ".PICT" or be prefaced with "PICT:".

Here is a small example that writes the entire screen to a file called "BACKGRND.PICT".

Pic.ScreenSave (0, 0, **maxx**, **maxy**, "BACKGRND.PICT")

Importing PCX Images in DOS OOT

Displaying a PCX image in DOS OOT is a two step process. First the PCX image must be converted to a format that Turing can read in directly (the TM2 format) and then the Turing program itself must read in the new image.

To convert the PCX image into TM2 format, Turing provides a program called PCX2TM2.EXE. This program reads in a PCX file and creates a file with the same name but with the extension ".TM2". Once this is done, the Turing program can call the **Pic.ScreenLoad** and **Pic.ScreenSave** procedures located in the **Pic** module.

The PCX format is used by PC-Paintbrush (a well known "paint" program). We decided to allow the interchange from PCX instead of one of the many other graphical standards because PCX has been around for a long time, it is supported by most draw and paint programs, and is available as a conversion format from many public domain and shareware format-conversion programs.

PCX2TM2

To convert the PCX image to a TM2 image, you use the PCX2TM2.EXE program. PCX2TM2.EXE is a DOS executable (i.e. you run it from the DOS prompt, rather than from inside Turing.) It has the following format:

pcx2tm2 {<options>} {<video mode>} <pcx file>

If you run PCX2TM2 without any command line arguments, it will ask for a command line. This allows PCX2TM2 to be used from Windows and menu programs that do not allow command line arguments.

The <option> is one of three:

-default This causes the TM2 file generated by PCX2TM2 to use the default color map. PCX2TM2 will attempt to remap the colors in the PCX

images color palette to provide the closest possible fidelity. The advantage of such an image is that it can be displayed without affecting any of the other colors on the display. If you are attempting to display two or more PCX images at once, this is especially important.

- debug** Provides some information about the PCX image before it is displayed.
- buffersize** This determines the size of the chunks in which the picture is saved. This number is the size of the takepic buffer in integers. The default value is 1,000, although you can set it to a maximum of 16,000. Note that the buffer is temporarily allocated on the stack when the image is displayed. You must have enough stack space to do this or the program will halt with a stack overflow message.

In general, the larger the buffer size, the faster the image will be displayed, but the more memory it will take.

The <video mode> is the mode in which the PCX image will be displayed. If none is specified, then PCX2TM2 determines the mode from the image. It is possible for it to guess incorrectly, although it will usually do a good job. If the video mode selected has a different number of colors than the image does, then PCX2TM2 will attempt to map the colors that the PCX image uses to the colors available in the video mode selected. Note that displaying a 256 color image in a 16 color screen mode usually produces a fairly muddy image. TM2 files are video mode specific. This means that if you display the PCX file in a particular video mode, the screen must be in the same video mode when **Pic.ScreenLoad** is called or the Turing program will give an error.

Specifying the video mode is one way to allow several PCX images of different resolutions to be displayed at once.

MODE	RESOLUTION	TURING MODE
cga	320 x 200 4-color CGA	graphics
16	320 x 200 16-color EGA	graphics:16
h16	640 x 200 16-color EGA	graphics:h16
ega	640 x 350 16-color EGA	graphics:ega
v2	640 x 480 2-color VGA	graphics:v2
vga	640 x 480 16-color VGA	graphics:vga
mcga	320 x 200 256-color MCGA	graphics:mcga
svga	640 x 480 256-color SVGA	graphics:svga
svga1	640 x 400 256-color SVGA	graphics:svga1
svga2	640 x 480 256-color SVGA	graphics:svga2
svga3	800 x 600 16-color SVGA	graphics:svga3
svga4	800 x 600 256-color SVGA	graphics:svga4
svga5	1024 x 768 16-color SVGA	graphics:svga5
svga6	1024 x 768 256-color SVGA	graphics:svga6

The last item is the actual name of the file to be converted. The output file will be the same name as the input file, but with a TM2 extension, hence "pcx2tm2 mypic.pcx" produces the output file "mypic.tm2".

One thing that you should be aware of is that PCX files can contain a palette. In other words, they can change the natural mapping of the screen colors (or color table). For example, an image of a forest might contain 200 different shades of green and no shades of blue whatsoever.

When such an image is translated by PCX2TM2, the palette information is stored with it unless the **-default** option is used. When the image is displayed using *Pic.ScreenLoad*, Turing reads in the color palette and changes the color palette used by the display to the one used by the PCX image. This means that if you are doing other graphics besides displaying the file, the colors in those graphics may suddenly change.

Note that when you use the **-default** option, the color fidelity of the image will be reduced. The forest scene that once had 200 shades of green may now use only the 30 shades of green provided by the default color palette.

If you are displaying more than one PCX image on the screen at the same time, the **-default** option is almost required. Otherwise when the second image is displayed, the color palette will be remapped a second time, causing the colors in the first image to be completely changed. Without the **-default** option, a forest scene displayed before a fire scene may have its entire color palette changed to shades of red when the fire image is displayed!

One limitation of PCX2TM2 is that it can not process 24-bit color images at this time. More to the point, Turing does not have any graphics modes that it could use to display such an image.

Pic.ScreenLoad

The **Pic.ScreenLoad** procedure has the following format:

Pic.ScreenLoad (*filename* : **string**, *x*, *y*, *mode* : **int**)

This procedure is the equivalent of **drawpic**, except that it draws from a file instead of an array. It will display the TM2 file, with the lower-left corner of the picture being (*x*, *y*). For full screen pictures, *x* and *y* should be 0. The *mode* parameter must be either *picCopy*, in which case the image displays over top of whatever is on the screen, or *picXor*, in which case the image is "xor"-ed with the underlying image on the screen.

If the procedure is unable to open or display the file it will not display an image. Instead **Error.Last** will be set to non-zero. It is a good idea to always check **Error.Last** to make certain that it is non-zero.

Here is a small example that displays the file "FOREST.TM2" at location (50, 50).

```
Pic.ScreenLoad ("FOREST.TM2", 50, 50, picCopy)
if Error.Last not= 0 then
    put "Error displaying image: ", Error.LastMsg
end if
```

Pic.ScreenLoad requires that either the file name end with ".TM2" or the file name be prefaced with "TM2:". For example, if the TM2 file was called "FOREST.PCT", then the **Pic.ScreenLoad** line would look like:

```
Pic.ScreenLoad ("TM2:FOREST.PCT", 50, 50, picCopy)
```

Pic.ScreenSave

The **Pic.ScreenSave** procedure has the following format:

```
Pic.ScreenSave (x1, y1, x2, y2 : int, filename : string)
```

This procedure is the equivalent of **takepic**, except it copies the screen to a file instead of an array. The section of the screen located in the rectangle specified by (*x1*, *y1*) - (*x2*, *y2*) is copied to a TM2 file located at *filename*. As with **Pic.ScreenLoad**, the file name must end with ".TM2" or be prefaced with "TM2:".

Here is a small example that writes the entire screen to a file called "BACKGRND.TM2".

```
Pic.ScreenSave (0, 0, maxx, maxy, "BACKGRND.TM2")
```


Chapter 7

Language Features

Alphabetic listing of all features of Turing.

abs absolute value function

Syntax	abs (<i>expn</i>)
Description	The abs function is used to find the absolute value of a number (the <i>expn</i>). For example, abs (-23) is 23.
Example	This program outputs 9.83. <pre>var x : real := -9.83 put abs (x) % Outputs 9.83</pre>
Details	The abs function accepts numbers that are either int 's or real 's. The type of the result is the same type as the accepted number. The abs function is often used to see if one number x is within a given distance d of another number y ; for example: <pre>if abs (x - y) <= d then ...</pre>
See also	predefined unit Math .

addr address of a variable

Dirty

Syntax	addr (<i>reference</i>)
Description	The addr attribute is used to find the integer address of a variable or non scalar constant. This is implementation-dependent. This address may be used in an indirection operation @.
Example	Set <i>a</i> to be the address of <i>x</i> . <pre>var x : real var a : addressint := addr (x)</pre>
Details	<p>The value of the address produced by addr is of type addressint, an integer type whose range is that of the underlying memory addresses.</p> <p>The concept of an address is implementation-dependent. For example, an optimizing compiler could determine that a variable does not require space because the program could be computed without the variable with no change in output. However, in most implementations, types have a predictable size and variables of that type occupy that number of bytes in memory.</p>
See also	the indirection operator @, cheat , <i>explicitIntegerConstant</i> (how to write hexadecimal constants), and pointer type (in particular unchecked pointer type). See also sizeof , which returns the size of a variable.

addressint type

Dirty

Syntax	addressint
Description	The addressint (address integer) type is an integer type whose range of value is the same as that of the underlying computer. This range is, by its nature, implementation-dependent. On 32-bit architectures, it is commonly the same range as nat4 (4-byte natural number).

Example Record *r* contains three fields, one of which has type **char**(28). Variable *a* is an integer whose range of values is the same as the addresses of the underlying computer. This assigns *B* to the seventh character of a record of type *r* which is assumed to be located at absolute address *a*.

```

type r :
  record
    i : int
    c28 : char ( 28 )
    c11 : char ( 11 )
  end record
var a : addressint      % An integer
...                      % a is assigned an integer value
                        r @ ( a ) . c28 ( 7 ) := 'B'      % Use indirection operator @

```

Details Although **addressint** is called an integer type, it is commonly equivalent to a natural type such as **nat4** (for 32-bit machines) or **nat2** (for 16-bit machines).

Be careful not to confuse **addressint** with pointer types. In low level languages such as assembler and C, addresses and pointers are the same. In Turing, however, a pointer is a high level concept that is more abstract than a machine address. A Turing pointer is a reference to an object, and the representation of this reference depends upon the implementation. In current Turing implementations, pointers (which are by default checked) are represented as a time stamp (a unique number) together with an address. The time stamp is used to make sure that the pointer actually locates an object. There are also **unchecked** pointers. An **unchecked** pointer's internal representation is a machine address. You can use type cheats (a dangerous feature) to translate between **addressint** and unchecked pointers. This is meaningful in current implementations.

See also the indirection operator @, **cheat**, *explicitIntegerConstant* (how to write hexadecimal constants), and **pointer** type (in particular **unchecked pointer** type). See also **addr**, which returns the address of a variable.

all all members of a set

Syntax *setType*Name (**all**)

Description Given a set type named *S*, the set of all of the possible elements of *S* is written *S* (**all**).

Example **type** *smallSet* : **set of** 0 .. 2

```
var x : smallSet := smallSet ( all )
```

% Set x contains elements 0, 1 and 2

See also **set** type for details about sets.

and operator

Syntax **A and B**

Description The **and** (boolean) operator yields a result of true if, and only if, both operands are true. The **and** operator is a short circuit operator. For example, if A is false in **A and B** then B is not evaluated.

Example

```
var success : boolean := false
var continuing := true      % The type is boolean
...
      continuing := continuing and success
```

Details The *continuing* variable is set to **true** if, and only if, both *continuing* and *success* are **true**. Since Turing uses short circuit operators, once *continuing* is **false**, *success* will not be looked at.

The **and** operator can also be applied to natural numbers. The result is the natural number that is the bit-wise **and** of the operands. See **nat** (natural number).

Example This masks out the everything but the lower two bytes of *number*.

```
var number : nat := 16#ABCD
var mask : nat := 16#FF
put number and mask      % Outputs 205 (CD16)
```

See also **boolean** (which discusses true/false values), *explicitTrueFalseConstant* (which discusses the values **true** and **false**), *precedence* and *expn* (expression).

anyclass the ancestor of all classes

Syntax	anyclass
Description	There is a predefined class called anyclass , which is the root of the expansion tree. All classes that do not have inherit lists are considered to be expansions of anyclass . The main purpose of anyclass is to allow pointers that can locate objects of any class.
Example	<p>Here is the declaration of a pointer <i>p</i> that can locate an object of any class.</p> <pre>var p : pointer to anyclass % Short form: var p : ^ anyclass var q : pointer to stack % Short form: var q : ^ stack new q % Create a stack object p := q % Legal because p's class % is an ancestor of q's class</pre> <p>Assuming <i>stack</i> is a class, this creates a <i>stack</i> object and places its location in <i>q</i> and <i>p</i>. The compiler will not allow a call to <i>stack</i>'s exported subprograms using <i>p</i> directly, as in:</p> <pre>p -> push (14) % ILLEGAL! anyclass has no operations</pre> <p>An assignment from <i>p</i> to <i>q</i> is legal, as in:</p> <pre>q := p % Checks that p locates a stack object (or descendant)</pre> <p>This implies a run time check to make sure that <i>p</i> locates an object that is a <i>stack</i> (or a descendant of a <i>stack</i>).</p> <p>Here is a way to call a subprogram exported from <i>stack</i> using <i>p</i>:</p> <pre>stack (p) . push (14) % Checks that p locates a stack object</pre> <p>This checks to see that <i>p</i> locates a <i>stack</i> object (or a descendant) before calling the <i>stack</i> operation <i>push</i>.</p>
Details	It is legal to create objects of the class called anyclass , but this is not of much use, because there is nothing you can do with these objects (they have no operations). It is legal to assign these objects to other objects of the same class (anyclass), although this accomplishes nothing.
See also	objectclass , which takes a class pointer and produces the class of the object located by the pointer. This is used for testing to determine the class of the object located by a pointer. See also class . See also export list, import list, inherit list, implement list and implement by list.

arctan arctangent function (radians)

Syntax **arctan** (*r* : **real**) : **real**

Description The **arctan** function is used to find the arc tangent of a value. The result is given in radians. For example, **arctan** (1) is $\pi / 4$.

Example This program prints out the arctangent of 0 through 3 in radians.

```
for i : 0 .. 12
  const arg := i / 4
  put "Arc tangent of ", arg, " is ",
    arctan ( arg ), " radians"
end for
```

Note The formulae for arcsin and arccos are:

$$\arcsin (x) = \mathbf{arctan} (\mathbf{sqrt} ((x * x) / (1 - (x * x))))$$

$$\arccos (x) = \mathbf{arctan} (\mathbf{sqrt} ((1 - (x * x)) / (x * x)))$$

See also the **arctand** function which finds the arc tangent of a value with the result given in degrees. (2π radians are the same as 360 degrees.)

See also predefined unit **Math**.

arctand arctangent function (degrees)

Syntax **arctand** (*r* : **real**) : **real**

Description The **arctand** function is used to find the arc tangent of an angle given in degrees. For example, **arctand** (0) is 0.

Example This program prints out the arctangent of values from 0 to 3 in degrees.

```
for i : 0 .. 12
  const arg := i / 4
  put "Arc tangent of ", arg, " is ",
    arctand ( arg ), " degrees"
end for
```

Note	<p>The formulae for arcsind and arccosd are:</p> $\text{arcsind}(x) = \text{arctand}(\text{sqrt}((x * x) / (1 - (x * x))))$ $\text{arccosd}(x) = \text{arctand}(\text{sqrt}((1 - (x * x)) / (x * x)))$
See also	<p>the arctan function which finds the arc tangent of a value with the result given in radians. (2π radians are the same as 360 degrees.)</p> <p>See also predefined unit Math.</p>

array type

Syntax	array <i>indexType</i> { , <i>indexType</i> } of <i>typeSpec</i>
Description	An array consists of a number of elements. The <i>typeSpec</i> gives the type of these elements. There is one element for each item in the (combinations of) range(s) of the <i>indexType</i> (s). In the following example, the array called <i>marks</i> consists of 100 elements, each of which is an integer.
Example	<pre> var marks : array 1 .. 100 of int ... var sum : int := 0 for i : 1 .. 100 % Add up the elements of marks sum := sum + marks (i) end for </pre>
Details	In the above example, <i>marks(i)</i> is the <i>i</i> -th element of the <i>marks</i> array. We call <i>i</i> the <i>index</i> or <i>subscript</i> of <i>marks</i> . In Turing, a subscript is surrounded by parentheses, not by square brackets as is the case in the Pascal language.
Example	<p>The <i>prices</i> array shows how an array can have more than one dimension. This array has one dimension for the year (1988, 1989 or 1990) and another for the month (1 .. 12). There are 36 elements of the array, one for each month of each year.</p> <pre> var price : array 1988 .. 1990, 1 .. 12 of int ... var sum : int := 0 for year : 1988 .. 1990 % For each year for month : 1 .. 12 % For each month sum := sum + price (year, month) end for end for </pre>

Details Each *indexType* must contain at least one item. The range 1 .. 0, for example, would not be allowed. Each index type must be a subrange of the integers, characters (the **char** type), or of an enumerated type, an (entire) enumerated type, the **char** type, the **boolean** type, or a named type which is one of these.

Arrays can also be declared in the form

```
var a : array 1 .. * of typeSpec := init ( ... )
```

The upper bound of *a* will be computed from the count of the initializing values. Both **var** and **const** arrays can be declared this way. An array variable/constant declared with "*" as an upper bound must have an initializing list. Only one dimensional arrays may be declared in this form.

Arrays can be assigned as a whole (to arrays of an equivalent type), but they cannot be compared.

An array can be initialized in its declaration using **init**. For details, see **var** and **const** declarations.

Example In this example, the size of the array is not known until run time.

```
var howMany : int
get howMany
var height : array 1 .. howMany of real
    ...read in all the elements of this array...
function total ( a : array 1 .. * of real ) : real
    var sum : int := 0
    for i : 1 .. upper ( a )
        sum := sum + a ( i )
    end for
    result sum
end total

    put "Sum of the heights is ", total ( height )
```

Details The ends of the range of a subscript are called the *bounds* of the array. If these values are not known until run time, the array is said to be *dynamic*. In the above example, *height* is a dynamic array. Dynamic arrays can be declared as variables, as in the case for *height*. However, dynamic arrays cannot appear inside other types such as records, and cannot be named types. Dynamic arrays cannot be assigned and cannot be initialized using **init**.

In the above example, **upper**(*a*) returns the size of *a*. See also **upper** and **lower**.

In the declaration of an array parameter, the upper bound can be given as an asterisk (*), as is done in the above example. This means that the upper bound is taken from that of the corresponding actual parameter (from *height* in this example).

You can have arrays of other types, for example arrays of record. If *R* is an array of records, then *R*(*i*).*f* is the way to access the *f* field of the *i*-th element of array *R*.

Details

Arrays can also be made resizeable. This is done using the **flexible** keyword. The declaration syntax is:

```
var name : flexible array indexType { , indexType } of typeSpec
```

The indices may have compile-time or run-time upper bounds (the lower bound must be compile-time). The upper bounds can be changed by using:

```
new name , newUpper1 {,newUpper2}
```

The existing array entries will retain their values, except that any index made smaller will have the corresponding array entries lost. Any index made larger will have the new array entries uninitialized (if applicable).

Additionally, the upper bound (both in the declaration and the **new** statement) can be made one less than the lower bound. This effectively makes an array that contains 0 elements. It can later be increased in size with another **new**.

In the current implementation (1999), with a multi-dimensional array with a non-zero number of total elements, it is a run-time error to change any but the first dimension (unless one of the new upper bounds is one less than the corresponding lower bound, giving 0 elements in the array) as the algorithm to rearrange the element memory locations has not yet been implemented.

Currently, only variables can be declared in this form. There is no flexible array parameter type, although a flexible array can be passed to an array parameter with "*" as the upper bound.

Example

In this example, the array is resized to fit the number of elements in the file.

```
function getLines (fileName : string) : int
  var f, numLines : int
  var line : string
  open : f, fileName, get
  numLines := 0
  loop
    exit when eof (f)
    get : f, line : *
    numLines += 1
  end loop
  close : f
  result numLines
end getLines

procedure readFile (var lines : array 1 .. * of string, fileName : string)
  var f : int
  var line : string
  open : f, fileName, get
  for i : 1 .. upper (lines)
    get : f, lines (i) : *
  end for

  close : f
end readFile
```

```

var lines : flexible array 1 .. 0 of string
new lines, getLines ("text.dat")
readFile (lines, "text.dat")
for i : 1 .. upper (lines)
    put lines (i)
end for

```

assert statement

Syntax	assert <i>trueFalseExpn</i>
Description	An assert statement is used to make sure that a certain requirement is met. This requirement is given by the <i>trueFalseExpn</i> . The <i>trueFalseExpn</i> is evaluated. If it is true, all is well and execution continues. If it is false, execution is terminated with an appropriate message.
Example	Make sure that <i>n</i> is positive. assert <i>n</i> > 0
Example	This program assumes that the <i>textFile</i> exists and can be opened, in other words, that the open will set the <i>fileNumber</i> to a non-zero stream number. If this is not true, the programmer wants the program halted immediately. <pre> var fileNumber : int open : fileNumber, "textFile", read assert fileNumber not= 0 </pre>
Details	In some Turing systems, checking can be turned off. If checking is turned off, assert statements may be ignored and as a result never cause termination.

assignability of expression to variable

Description A value, such as 24, is assignable to a variable, such as *i*, if certain rules are followed. These rules, given in detail below, are called the *assignability* rules. They must be followed in assignment statements as well as when passing values to non-**var** parameters.

Example

```
var i : int
i := 24           % 24 is assignable to i

var width : 0 .. 319
width := 3 * i    % 3 * i is assignable to width

var a : array 1 .. 25 of string
a ( i ) := "Ralph" % "Ralph" is assignable to a(i)

var name : string ( 20 )
name := a ( i )   % a(i) is assignable to name

...
var b : array 1 .. 25 of string
b := a           % Array a is assignable to b

type personType :
  record
    age : int
    name : string ( 20 )
  end record
var r, s : personType
...
s := r           % Record r is assignable to s
```

Details The expression on the right of := must be *assignable* to the variable on the left. An expression passed to a non-**var** parameter must be assignable to the corresponding parameter.

An expression is defined to be *assignable* to a variable if the two *root* types are *equivalent* or if an integer value is being assigned to a **real** variable (in which case the integer value is automatically converted to **real**). Two types are considered to be equivalent if they are essentially the same type (see *equivalence* for the detailed definition of this term).

In most cases a *root* type is simply the type itself. The exceptions are subranges and strings. The *root* type of a subrange, such as 0 .. 319, is the type of its bounds (**int** type in this example). The *root* type of a string, such as the type **string**(9), is the most general string type, namely **string**.

When a subrange variable, such as *width*, is used as an expression, for example on the right side of an assignment statement, its type is considered to be the *root* type (integer in this case) rather than the subrange. When an expression is assigned to a subrange variable such as *width*, the value ($3*i$ in this example) must lie in the subrange. Analogously, any string variable used in an expression is considered to be of the most general type of string. When a string value is assigned to a string variable, its length must not exceed the variable's maximum length.

Turing's assignability rule applies to characters and strings in this way. A **char** value can be assigned (or passed to a non **var** parameter) with automatic conversion to a **char**(1) variable and vice versa. String values of length 1 can be assigned to **char** variables. Character, that is **char**, values can be assigned to string variables, yielding a string of length 1. String values of length *n* are assignable with automatic conversion to **char**(*n*) variables. Values of type **char**(*n*) can be assigned with automatic conversion to **string** variables.

Turing's assignability rule applies to pointers to classes in this way. A pointer that locates an object created as class *E*, can be assigned to a pointer to class *B* only if *B* is an ancestor of (or the same as) *E*. For example, a pointer to an object that is a *stackWithDepth* can be assigned to a pointer to *stack*, where *stackWithDepth* is a child of *stack*, but not vice versa. The pointer **nil** can be assigned to any pointer variable, but the value **nil**(*C*) can only be assigned to a pointer to an ancestor of *C*.

Objects of classes can be assigned to each other only if both were created as the same class.

assignment statement

Syntax An *assignmentStatement* is:

variableReference := *expn*

Description An assignment statement calculates the value of the expression (*expn*) and assigns that value to the variable (*variableReference*).

Example

```

var i : int
i := 24                % Variable i becomes 24
var a : array 1 .. 25 of string
a ( i ) := "Ralph"    % The i-th element of a becomes "Ralph"
...
var b : array 1 .. 25 of string
      b := a           % Array b becomes (is assigned) array a

```

Details	<p>The expression on the right of <code>:=</code> must be <i>assignable</i> to the variable on the left. For example, in the above, any integer value, such as 24, is assignable to <i>i</i>, but a real value such as 3.14 would not be assignable to <i>i</i>. Entire arrays, records and unions can be assigned. For example, in the above, array <i>a</i> is assigned to array <i>b</i>. See <i>assignability</i> for the exact rules of allowed assignments.</p> <p>You cannot assign a new value to a constant (const).</p> <p>There are short forms that allow you to write assignment statements more compactly. For example,</p> $i := i + 1$ <p>can be shortened to</p> $i += 1$ <p>In Turing, there are short forms for combining <code>+</code>, <code>=</code> and <code>*</code> with assignment. For example, <code>i *= 2</code> doubles <i>i</i>.</p> <p>There are also short forms to allow any binary operator to be combined with assignment. For example, <code>i shl= 2</code> shifts <i>i</i> by 2 to the left.</p>
---------	--

begin statement

Syntax	<p>A <i>beginStatement</i> is:</p> <pre> begin <i>statementsAndDeclarations</i> end </pre>
Description	<p>A begin statement limits the scope of declarations made within it to the confines of the begin/end block. In Turing, begin is rarely used, because declarations can appear wherever statements can appear, and because every structured statement such as if ends with an explicit end.</p>
Example	<pre> begin var bigArray : array 1 .. 2000 of real ... bigArray will exist only inside this begin statement... end </pre>
Details	<p>In Pascal programs, begin statements are quite common because they are required for grouping two or more statements, for example, to group the statements that follow then. This is not necessary in Turing as where ever you can write a single statement, you can also write several statements.</p>

bind declaration

Syntax	<p>A <i>bindDeclaration</i> is:</p> $\mathbf{bind} \ [\mathbf{var}] \ id \ \mathbf{to} \ variableReference$ $\{ \ , \ [\mathbf{var}] \ id \ \mathbf{to} \ variableReference \}$
Description	<p>The bind declaration creates a new name (or names) for a variable reference (or references). You are allowed to change the named item only if you specify var. You can also bind to named non scalar constants. While <i>variableReference</i> is bound it does not disappear in the scope.</p>
Example	<p>Rename the <i>n</i>-th element of array <i>A</i> so it is called <i>item</i> and then change this element to 15.</p> $\mathbf{bind} \ \mathbf{var} \ item \ \mathbf{to} \ A \ (\ n \)$ $item := 15$
Details	<p>The scope of the identifier (<i>item</i> above) begins with the bind declaration and lasts to the end of the surrounding program or statement (or to the end of the surrounding part of a case or if statement). During this scope, a change to a subscript (<i>n</i> above) that occurs in the variable reference does not change the element to which the identifier refers.</p> <p>You are not allowed to use bind at the outermost level of the main program (except nested inside statements such as if) or at the outermost level in a module.</p> <p>You can also optionally use the register keyword to request that the bind be done using a machine register. The syntax for <i>bindDeclaration</i> is actually:</p> $\mathbf{bind} \ [\mathbf{var}] \ [\mathbf{register}] \ id \ \mathbf{to} \ variableReference$ $\{ \ , \ [\mathbf{var}] \ [\mathbf{register}] \ id \ \mathbf{to} \ variableReference \}$ <p>In the current (1999) implementation, programs are run interpretively using pseudo-code and the register keyword is ignored.</p>

bits extraction

Syntax	$\mathbf{bits} \ (\ expn, \ subrange \)$
--------	--

Description	The bits operator is used to extract a sequence of bits from a natural (non-negative) number expression. The bits are numbered from right to left as 0, 1, 2 ...
Example	<p>Set bits 2 and 1 (third and second from the right) in the variable <i>d</i> to be 01. We first set <i>b</i> to be the bit string 1100.</p> <pre> type T12 : 1 .. 2 % Use to specify bit range var d : nat2 := 2#1100 % Two byte natural number % At this point bits(d, T12) = 2#10 bits (d, T12) := 2#01 % At this point d = 2#1010 </pre>
Example	<p>Set bit 7 in variable <i>n</i> to be 1. As a result, <i>n</i> will equal 2#10000000.</p> <pre> var n : nat1 := 0 % A one byte variable set to zero bits (n, 7) := 1 % n now contains the pattern 10000000 </pre>
Details	<p>The form of <i>subrange</i> must be one of:</p> <ul style="list-style-type: none"> (a) typeSpec % Subrange type (b) compileTimeIntegerExpression <p>In form (a) the subrange type specifies a range from <i>L</i> to <i>M</i> (for <i>least</i> and <i>most</i> significant). This is a little confusing because the subrange is written <i>L</i> .. <i>M</i> with <i>L</i> on the left and <i>M</i> on the right, but in a number, the least significant bit is on the right and the most significant is on the left. The subrange type can be either the name of a type, for example <i>T12</i>, or an explicit subrange, for example 3 .. 7. The values in the explicit subrange must be compile time values.</p> <p>Form (b) represents the range <i>n</i> .. <i>n</i> where <i>n</i> is the non-negative value of the expression. In other words, both <i>L</i> and <i>M</i> equal <i>n</i>. The expression can be any non-negative integer value or natural number value.</p> <p>If the expression <i>expn</i> is a variable reference, the bits operation can be assigned to, but cannot be passed to, a var parameter. For example, in the above, bits (<i>d</i>, <i>T12</i>) has the value 2#01 assigned to it. For this assignment to be allowed, the expression <i>expn</i> must be a natural number type (nat, nat1, nat2 or nat4).</p>
See also	<i>explicitIntegerConstant</i> (for description of constants such as 16#FFFF) and the following functions that convert one type to another in a machine-independent manner: ord , chr , intstr , strint , natstr , and strnat . See also shr and shl (shift right and left).

body declaration

Syntax	<p>A <i>bodyDeclaration</i> is one of:</p> <ul style="list-style-type: none">(a) body procedure <i>procedureId</i> <i>statementsAndDeclarations</i> end procedureId(b) body function <i>functionId</i> <i>statementsAndDeclarations</i> end functionId(c) body procedure <i>id</i> [(<i>paramDeclaration</i> {, <i>paramDeclaration</i> })] <i>statementsAndDeclarations</i> end id(d) body function <i>id</i> [([<i>paramDeclaration</i> {, <i>paramDeclaration</i> }])] : <i>typeSpec</i> <i>statementsAndDeclarations</i> end id
Description	<p>A body declaration is used to resolve either a forward subprogram or a deferred subprogram.</p> <p>You declare a procedure or function forward when you want to define its header but not its body. This is the case when one procedure or function calls another, which in turn calls the first. This situation is called <i>mutual recursion</i>. The use of forward is necessary in this case because every item must be declared before it can be used. The forward declaration must be followed by a body declaration for the same procedure or function. For details, see forward declarations.</p> <p>When a procedure or function in a class is declared to be deferred (or simply exported from the class), it can be resolved or <i>overridden</i> afterward by giving its body further down in that class or in descendant classes. The overriding procedure must use the keyword body. See class or "implement by" for examples.</p>
Details	<p>You can specify the parameter and return values of the subprogram in the body declaration. However, the names and types of the parameters and return values must match the initial declaration exactly, or a warning results and the parameter list and return values from the body declaration are ignored.</p>

Example The example given here is part of a complete Turing program that includes an explanation of **forward** declarations.

```
var token : string
forward procedure expn ( var eValue : real )
    import forward term, var token
... other declarations appear here ...
body procedure expn
    var nextValue : real
    term (eValue )           % Evaluate t
    loop                     % Evaluate { + t}
        exit when token not= "+"
        get token
        term (nextValue)
        eValue := eValue + nextValue
    end loop
end expn
```

Details The syntax of a *bodyDeclaration* presented above has been simplified by omitting the optional result identifier, **import** list, **pre** and **post** condition and **init** clause. See **procedure** and **function** declarations for descriptions of these omissions.

The "function" or "procedure" token in the declaration is now optional. In other words the following code fragment is legal

```
forward procedure p
...
body p
...
end p
```

boolean true-false type

Syntax **boolean**

Description The **boolean** type is used for values that are either **true** or **false**. These true-false values can be combined by various operators such as **or** and **and**.

Example

```
var success : boolean := false
var continuing := true        % The type is boolean
...
success := mark >= 60
continuing := success and continuing
```

if continuing then ...

Details	<p>This type is named after the British mathematician, George Boole, who formulated laws of logic.</p> <p>The operators for true and false are and, or, xor, =>, and not. For two true/false values <i>A</i> and <i>B</i>, these operators are defined as follows:</p> <ul style="list-style-type: none"><i>A</i> and <i>B</i> is true when both are true<i>A</i> or <i>B</i> is true when either or both are true<i>A</i> xor <i>B</i> is true when either but not both are true<i>A</i> => <i>B</i> (<i>A</i> implies <i>B</i>) is true when both are true or when <i>A</i> is falsenot <i>A</i> is true when <i>A</i> is false <p>The and operator has higher precedence than or, so <i>A</i> or <i>B</i> and <i>C</i> means <i>A</i> or (<i>B</i> and <i>C</i>).</p> <p>The operators or, and and => are short circuit operators. For example, if <i>A</i> is true in <i>A</i> or <i>B</i>, <i>B</i> is not evaluated.</p>
Details	The boolean type can be used as an index to an array.
Example	<p>Declaration of an array with boolean index.</p> <pre>var a : array boolean of int a (false) := 10 a (true) := 20</pre>
Details	<p>The put and get semantics allow put's and get's of boolean values. true values will be output as "true" and false values will be output as "false".</p> <p>The only legal input values are "true" and "false", which are case sensitive.</p>
See also	<i>explicitTrueFalseConstant</i> (which discusses the values true and false), <i>precedence</i> and <i>expn</i> (expression).

break debugger pause statement

Syntax	break
Description	<p>On systems with a debugger (currently WinOOT and MacOOT), the environment "pauses" when execution reaches the break statement. While "pausing" is environment specific, in both WinOOT and MacOOT, the program stops execution until the user presses the "Continue" button. While paused, the program variables can be inspected, stack traces done, etc.</p>

Example

```
for i : 1 .. 100
  put i
  break
end for
```

buttonchoose switch mouse modes

Syntax **buttonchoose** (*choice* : **string**)

Description The **buttonchoose** procedure is used to change the mode of the mouse. In Turing, the mouse can either be in "*single-button mode*" or in "*multi-button mode*". In "*single-button mode*" the mouse is treated as a one button mouse. A button is considered pressed when any button is pressed and released only when all buttons have been released.

In Turing, the mouse starts in "*single-button mode*".

The parameter *choice* can be one of "singlebutton", "onebutton" (which switch the mouse into "*single-button mode*") or "multibutton" (which switches the mouse into "*multi-button mode*").

Example A program that displays the status of the mouse at the top left corner of the screen.

```
buttonchoose ("multibutton")
var x, y, button, left, middle, right : int
mousewhere (x, y, button)
left := button mod 10            % left = 0 or 1
middle := (button - left) mod 100 % middle = 0 or 10
right := button - middle - left   % right = 0 or 100
if left = 1 then
  put "left button down"
end if
if middle = 10 then
  put "middle button down"
end if
if right = 100 then
  put "right button down"
end if
```

Details Under DOS, the mouse does not start initialized. When the first mouse command is received (**mousehide**, **mousethrow**, **mousewhere**, **buttonmoved**, **buttonwait**, **buttonchoose**) Turing attempts to initialize the mouse. If successful, the mouse cursor will appear at that time.

Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the **autoexec.bat** or **config.sys** files).

See also **buttonmoved** and **buttonwait** to get mouse events saved in a queue. See also **mousewhere** to get the current status of mouse button(s).
See also predefined unit **Mouse**.

buttonmoved has a mouse event occurred

Syntax **buttonmoved** (*motion* : string) : boolean

Description The **buttonmoved** function indicates whether there is a mouse event of the appropriate type on the mouse queue. Events are either "up", "down", "updown" or "downup" events (although the "downup" and "updown" are the same event).

The parameter *motion* must be one of "up", "down", "updown" or "downup". If an event of the type requested is in the queue, **buttonmoved** returns **true**. If the event is not in the queue, then **buttonmoved** returns **false**.

In "*single-button mode*" (where the mouse is treated like a one-button mouse), a "down" event occurs whenever all the buttons are up and a button is pressed. An "up" event takes place when the last button is released so that no buttons remain pressed.

In "*multi-button mode*", a "down" event occurs whenever any button is pressed, and an "up" event occurs whenever any button is released.

Example This program draws random circles on the screen until the user clicks the mouse button, whereupon it starts drawing random boxes. Clicking the mouse button switches between the two.

```
var circles: boolean := true
loop
  var x, y, radius, clr: int
  if buttonmoved ("down") then
    var buttonnumber, buttonupdown : int
    buttonwait ("down", x, y, buttonnumber, buttonupdown)
    circles := not circles
  end if
  randint (x, 0, maxx)
  randint (y, 0, maxy)
  randint (radius, 0, 100)
```

```

randint (clr, 0, maxcolor)
if circles then
    drawfilloval (x, y, radius, radius, clr)
else
    drawfillbox (x, y, x + radius, y + radius, clr)
end if
end loop

```

Example This is an example demonstrating how to check for both character and mouse input at the same time.

```

var ch : string (1)
var x, y, btnnum, btnupdown : int
loop
    if hasch then
        getch (ch)
        locate (1, 1)
        put "The character entered is a: ", ch
    end if
    if buttonmoved ("down") then
        buttonwait ("down", x, y, btnnum, btnupdown)
        locate (1, 1)
        put "The button was clicked at position: ", x, ", ", y
    end if
end loop

```

Details **buttonmoved** can be thought of as the mouse equivalent of **hasch** in that they both check for something in a queue and both return immediately.

Under DOS, the mouse does not start initialized. When the first mouse command is received (**mousehide**, **mousethrow**, **mousewhere**, **buttonmoved**, **buttonwait**, **buttonchoose**), Turing attempts to initialize the mouse. If the initialization is successful, the mouse cursor will appear.

Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the **autoexec.bat** or **config.sys** files).

See also **buttonmoved** to get mouse events saved in the queue. See also **buttonchoose** to switch between "single-button mode" and "multi-button mode".

See also predefined unit **Mouse**.

buttonwait get a mouse event procedure

Syntax	buttonwait (<i>motion</i> : string , var <i>x</i> , <i>y</i> , <i>buttonnumber</i> , <i>buttonupdown</i> : int)
Description	<p>The buttonwait procedure gets information about a mouse event and removes it from the queue.</p> <p>The parameter <i>motion</i> must be one of "up", "down", "updown" or "downup". If an event of the type requested is in the queue, buttonwait returns instantly. If there isn't such an event, buttonwait waits until there is one and then returns (much like getch handles keystrokes).</p> <p>In "single-button mode" (where the mouse is treated like a one-button mouse), a "down" event occurs whenever all the buttons are up and a button is pressed. An "up" event takes place when the last button is released so that no buttons remain pressed.</p> <p>In "multi-button mode", a "down" event occurs whenever any button is pressed, and an "up" event occurs whenever any button is released.</p> <p>The parameters <i>x</i> and <i>y</i> are set to the position of the mouse cursor when the button was pressed. The parameter <i>buttonnumber</i> is set to 1 when in "single-button mode". In "multi-button mode", it is set to 1 if the left button was pressed, 2 if the middle button was pressed, and 3 if the right button was pressed. The parameter <i>buttonupdown</i> is set to 1, if a button was pressed and 0 if a button was released.</p>
Example	<p>This program draws lines. It starts a line where the user presses down and continues to update the line while the mouse button is held down. When the button is released, the line is permanently draw and the user can draw another line.</p> <pre>var <i>x</i>, <i>y</i>, <i>buttonnumber</i>, <i>buttonupdown</i>, <i>buttons</i> : int var <i>nx</i>, <i>ny</i> : int loop buttonwait ("down", <i>x</i>, <i>y</i>, <i>buttonnumber</i>, <i>buttonupdown</i>) <i>nx</i> := <i>x</i> <i>ny</i> := <i>y</i> loop drawline (<i>x</i>, <i>y</i>, <i>nx</i>, <i>ny</i>, 0) % Erase previous line exit when buttonmoved ("up") mousewhere (<i>nx</i>, <i>ny</i>, <i>buttons</i>) drawline (<i>x</i>, <i>y</i>, <i>nx</i>, <i>ny</i>, 1) % Draw line to position end loop buttonwait ("up", <i>nx</i>, <i>ny</i>, <i>buttonnumber</i>, <i>buttonupdown</i>) drawline (<i>x</i>, <i>y</i>, <i>nx</i>, <i>ny</i>, 2) % Draw line to final position</pre>

end loop

Example This is an example demonstrating how to check for both character and mouse input at the same time.

```
var ch : string (1)
var x, y, btnnum, btnupdown : int
loop
  if hasch then
    getch (ch)
    locate (1, 1)
    put "The character entered is a: ", ch
  end if
  if buttonmoved ("down") then
    buttonwait ("down", x, y, btnnum, btnupdown)
    locate (1, 1)
    put "The button was clicked at position: ", x, ", ", y
  end if
end loop
```

Details **buttonwait** can be thought of as the mouse equivalent of **getch** in that they both read something in a queue and both wait until they get the thing they're looking for.

Under DOS, the mouse does not start initialized. When the first mouse command is received (**mousehide**, **mousethrow**, **mousewhere**, **buttonmoved**, **buttonwait**, **buttonchoose**), Turing attempts to initialize the mouse. If the initialization is successful, the mouse cursor will appear.

Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the **autoexec.bat** or **config.sys** files).

See also **buttonwait** to see if an appropriate event is in the queue. See also **buttonchoose** to switch between "single-button mode" and "multi-button mode".

See also predefined unit **Mouse**.

case selection statement

Syntax A *caseStatement* is:

```
case expn of
  { label compileTimeExpn {, compileTimeExpn } :
    statementsAndDeclarations }
```

[**label** :
statementsAndDeclarations]
end case

Description A **case** statement is used to choose among a set of statements (and declarations). One set is chosen and executed and then execution continues just beyond **end case**.

The expression (*expn*) following the keyword **case** is evaluated and used to select one of the alternatives (sets of declarations and statements) for execution. The selected alternative is the one having a label value equaling the case expression. If none are equal and there is a final **label** with no expression, that alternative is selected.

Example Output a message based on value of mark.

```
case mark of
  label 9, 10 : put "Excellent"
  label 7, 8 :  put "Good"
  label 6 :    put "Fair"
  label :      put "Poor"
end case
```

Example Output a message based on value of name.

```
case name of
  label "horse", "cow" : put "Farm animal"
  label "tiger", "lion" : put "Jungle animal"
  label "cat", "dog" :   put "Pet"
  label :                put "Unknown animal"
end case
```

Details The case expression is required to match one of the labels. If it does not, there must be a final **label** with no expression. Label expressions must have values known at compile time. All label values must be distinct. The case expression and the label values must have the same equivalent type, which must be an integer, **char**, **boolean**, an **enum** type or strings.

catenation (+) joining together strings

Syntax A *catenation* is:

stringExpn + *stringExpn*

Description Two strings (*stringExprs*), **char** or **char**(*n*) values can be joined together (catenated) using the + operator.

Example

```
var lastName, wholeName : string
lastName := "Austere"
wholeName := "Nancy" + " " + lastName
    % The three strings Nancy, a blank and Austere
    % catenated together to make the string
    % "Nancy Austere". This string becomes the
    % value of wholeName
```

Details The length of a string catenation is limited to 255 characters.

Catenation is sometimes called *concatenation*.

Catenation can also be applied to **char** and **char**(*n*) values. See **char** and **char**(*n*). If either operand, *s* or *t* in *s* + *t*, is a **string** or a dynamic **char**(*n*) (length not known at compile time), the result type is **string**. Otherwise (when both *s* and *t* are **char** or non-dynamic **char**(*n*)) the result type is **char**(*n*).

The result of catenation is considered to be a compile time value if both operands are compile time values.

If both operands have the type **char** or **char**(*n*) neither of which is a dynamic **char**(*n*), the result is of type **char**(*n*), which is also of a non dynamic type. This allows the creation of very long **char**(*n*) values that can effectively span line boundaries using catenation to join lines. If either operand is a dynamic type or a string type, the catenation produces a string, whose length is limited to 255 characters.

See also *substrings* (for separating a strings into parts), **repeat** (for making repeated catenations), **string** type, **length**, and **index** (to determine where one string is located inside another).

See also **string**, **char**, **char**(*n*), `explicitStringConstant`, `explicitCharConstant`, `substring` and **length**.

ceil real-to-integer function

Syntax **ceil** (*r* : **real**) : **int**

Description Returns the smallest integer greater than or equal to *r*.

Details	The ceil (ceiling) function is used to convert a real number to an integer. The result is the smallest integer that is greater than or equal to <i>r</i> . In other words, the ceil function rounds up to the nearest integer. For example, ceil (3) is 3, ceil (2.25) is 3 and ceil (-8.43) is -8.
See also	See also the floor and round functions.

char type

Syntax	char
Description	Each variable whose type is a char contains a single character, such as the letter <i>A</i> , the digit 3 or the special character <i>&</i> .
Example	<p>Count characters until a period is found. Character <i>c</i> is read using a get statement and is compared to the explicit character constant '.'.</p> <pre> var c : char var counter := 0 loop exit when eof get c % Read a single character exit when c = '.' % Single quotes for char constant counter := counter + 1 end loop put counter, " characters before the period"</pre>
Example	<p>Count capital letters. This example illustrates the use of the char type as the subscript type for the <i>frequency</i> array, the use of character variable <i>c</i> as a subscript, and the use of <i>d</i> as a for counter that ranges across the letters A to Z.</p> <pre> var frequency : array 'A' .. 'Z' of nat for d : 'A' .. 'Z' frequency (d) := 0 end for loop % Tabulate use of capital letters exit when eof var c : char get c % Read one character if c >= 'A' and c <= 'Z' then frequency (c) := frequency (c) + 1 end if end loop for d : 'A' .. 'Z' % Print frequency of capital letters put d, " ", frequency (d)</pre>

end for

Details

The type **string** (or **char(*n*)**) is used instead of **char** when more than one character needs to be stored, such as the string of characters *Henry*. Unless the program needs to be quite efficient, it is usually easier to use the **string** type. See also the **char(*n*)** type, which always stores exactly *n* characters.

The **char** type differs from the **string(1)** type in the following way: **char** always represents exactly one character, while **string(1)** can represent either the null string or a string containing one character. The **char** type is similar to the **char(1)** type in that both contain at most one character.

The **char** type is an index type and can be used, for example, as subscripts, **for** ranges and **case** labels. For example, this declaration

```
var charCounts : array char of int
```

creates an array whose subscripts are characters.

The **char** type is a scalar type, which implies that its parameters are passed by value, instead of by reference (which is the case for **char(*n*)** and **string**).

Values of the **char** type can be assigned and they can be compared for both equality and ordering. Explicit **char** constants are written as a character surrounded by single quotes, for example, 'A'. For details, including how to write control characters, see *explicitCharConstant*.

Characters can be read and written by **get** and **put** statements.

There are 256 **char** values, corresponding to the distinct patterns in an 8-bit byte. This allows the patterns *eos* (internal value 0) and *uninitchar* (internal value 128) to be **char** values (these patterns are not allowed in the **string** type; see the **string** type). All 256 patterns are used, so there is no pattern left to be the "uninitialized value". Uninitialized checking is not done for the **char** type.

The **ord** and **chr** functions convert between the **char** values and their corresponding numeric representation in a byte. See **ord** and **chr**.

In general, you can freely intermix the values of the types **char**, **char(*n*)** and **string**. This means that catenation (+), comparisons, **length** and substrings can be applied to any of these types. See **char(*n*)** for details about conversions between **char**, **char(*n*)** and **string**.

char(*n*) type

Syntax

```
char ( numberOfCharacters )
```

Description	Each variable whose type is a char (<i>n</i>) contains exactly <i>n</i> characters.
Example	<p>Canadian postal codes contain six characters, for example, M4V 1Y9. This is represented in a char(6) variable:</p> <pre>var postalCode : char (6) := 'M4V1Y9'</pre>
Details	<p>Explicit constants for the char(<i>n</i>) type use single quotes as in 'M4V1Y9', as opposed to explicit string constants which use double quotes, as in "Nancy". A single character single quoted character, such as 'A', is considered to have the type char instead of char(<i>n</i>), but since these two types can be assigned to each other and compared to each other, this fact has little consequence.</p> <p>The type char(<i>n</i>) is generally more difficult to use than the string type, which is favored for most simple programs. The type char(<i>n</i>) has the advantage that it is efficient in terms of both space and time. In particular, it is represented as <i>n</i> bytes in the computer's memory. By contrast, the string type must use extra space (a trailing zero byte in current implementations) to represent the current length and allocates space for the maximum value it can hold.</p> <p>The form of <i>numberOfCharacters</i> is one of:</p> <p>(a) <i>expn</i> % Integer value</p> <p>(b) * % Only in subprogram parameters</p> <p>The first form determines <i>n</i>. If the expression is a run time value, the type is considered to be <i>dynamic char</i>(<i>n</i>). The value of <i>n</i> must be at least 1. The second form is used only for subprogram parameters and uses the length of the actual parameter. This too, is considered to be a <i>dynamic char</i>(<i>n</i>) type. Dynamic char(<i>n</i>) types can only be passed to char(*) parameters. Dynamic char(<i>n</i>) types have the same restrictions as dynamic arrays. This implies they cannot be assigned as a whole and cannot appear in record and union types.</p> <p>An implementation may impose a limit, recommended to be at least 32767, on the length <i>n</i>.</p> <p>Values of the char(<i>n</i>) type can be assigned and they can be compared for both equality and for ordering, but only if they have the same length and they are not dynamic (i.e. the length must be known at compile time).</p> <p>Values of the char(<i>n</i>) type can be read and written by get and put statements.</p> <p>The char(<i>n</i>) type is a nonscalar, which implies that its parameters are always passed by reference (by means of an implicit pointer).</p> <p>As is true for the char type, all 256 possible values of an 8-bit byte are allowed for each character in char(<i>n</i>) type. There is no pattern left to be used for the "initialized value", so there is no uninitialized checking for char(<i>n</i>).</p>

In general, you can freely intermix the values of the types **char**, **char(*n*)** and **string**. This means that catenation (+), comparisons, **length** and substrings can be applied to any of these types. See **catenation** and **substring**. If two non dynamic **char(*n*)** values (or **char** values) are catenated, the result is a **char(*n*)** value. If either are dynamic, it is a **string** value. This implies that very long **char(*n*)** values can be created by catenating them together, for example to initialize a **char(*n*)** variable.

A **char** value can be assigned (or passed to a non **var** parameter) with automatic conversion to a **char(1)** variable and vice versa. String values of length 1 can be assigned to **char** variables. Character (**char**) values can be assigned to string variables, yielding a string of length 1. String values of length *n* are assignable with automatic conversion to **char(*n*)** variables. Values of type **char(*n*)** can be assigned with automatic conversion to **string** variables.

When comparing two **char(*n*)** values, as in *s > t*, if both are non-dynamic and of the same length, they are compared without converting to strings. If either are dynamic, they are converted to strings and then compared.

See also the **char** type which is much like **char(1)**. See also the **string** type.

cheat type cheating Dangerous

Syntax	<p>A <i>typeCheat</i> is one of:</p> <ul style="list-style-type: none"> (a) cheat (<i>targetType</i>, <i>expn</i> [: <i>sizeSpec</i>]) (b) # expn (c) <i>id</i> : cheat <i>typeSpec</i>
Description	<p>A type cheat interprets the representation (bits) of one type as another type. Type cheats are dirty (machine-dependent) and sometimes dangerous (arbitrary corruption) and should be used only by programmers who know the underlying computer representation of values.</p> <p>Form (b) is a short form type cheat in which the target type is a natural number.</p> <p>Form (c) is used as a parameter in a subprogram declaration. It causes whatever is passed in to the parameter to be interpreted as <i>typeSpec</i>.</p>
Example	<p>The character 'B' is assigned to variable <i>i</i>, whose type is considered to be char (although it is really int1).</p> <pre> var i : int1 % One byte integer cheat (char, i) := 'B' </pre>

This assignment is equivalent (on byte oriented computers) to either of the following:

```
i := cheat ( int1, 'B' )
      i := ord ( 'B' )
```

Details The form of *targetType* must be one of:

- (a) [*id* .] *typeId*
- (b) **int**, **int1**, **int2** or **int4**
- (c) **nat**, **nat1**, **nat2** or **nat4**
- (d) **boolean**
- (e) **char** [(*numberOfCharacters*)]
- (f) **string** [(*maxLength*)]
- (g) **addressint**

In form (a) the beginning identifier *id* must be the name of a module, monitor or class that exports the *typeId*. Each of *numberOfCharacters* and *maxLength* must be compile time integer expressions.

If the *expn* in a type cheat is a variable reference and the *sizeSpec* is omitted, the type cheat is considered to be a variable whose type is *targetType*. This allows, for example, the type cheat to be assigned to, as in:

```
cheat ( char, i ) := 'B'
```

If the *expn* is a value that is not a variable reference, or if *sizeSpec* is present, the type cheat is an expression value whose type is *targetType*.

The *sizeSpec* is a compile time integer expression giving the size of the *expn*'s value. It can be specified only for integer or natural number values (where it must be 1, 2 or 4) or real values (where it must be 4 or 8).

A type cheat is carried out in two steps. The first step converts the value if necessary to the size given by *sizeSpec*. The second step, which involves no generated code, interprets the value as the target type.

The prefix operator # is a short form for a class of type cheats. It interprets its argument as a natural number. In general, # *expn* is the same as **cheat** (**nat***n*, *expn*) where *n* is determined as follows. If the *expn* is a variable or expression of size 1, 2 or 4, *n* is the size of the item, otherwise *n* is 4.

Example Set the second character of *d* so it has the numeric representation 24. In general, if *c* is a character, then #*c* = **ord**(*c*). Note that #*c* can have a number value assigned to it, but **ord**(*c*) cannot.

```
var d : char ( 3 )
      #d ( 2 ) := 24                      % Same as d(2) := chr(24)
```

Example The notation 16#FFFF means FFFF in base 16, which is 32767 in base 10 and is 16 1's in a row in base 2. This same pattern is the two's complement representation of the value -1 in a 2-byte integer.

```
var i : int2
      #i := 16#FFFF                      % Equivalent to i := -1
```

Example The following example prints out a string located at address `myAddr`.

```
procedure PrintString (str : cheat string)
  put str
end PrintString

var myAddr : addressint
...
PrintString (myAddr) % myAddr will be treated as a string
```

Details An implementation may prohibit certain type cheats. Memory alignment requirements may render some type cheats unfeasible. It is dangerous to consider a value to have a *targetType* larger than the value's type. An implementation may prohibit certain type cheats on **register** scalar items.

See also *explicitIntegerConstant* (for description of constants such as 16#FFFF) and the following functions that convert one type to another in a machine-independent manner: **ord**, **chr**, **intstr**, **strint**, **natstr**, and **strnat**.

checked compiler directive

Description Unchecked means that certain run time tests, which take place by default, can be eliminated, usually to make the program more efficient at the risk of unreliability. The keyword **checked**, used as a statement, requests that the disabling of checking, previously requested by the keyword **unchecked**, be re-enabled. See **unchecked** for details and an example.

chr integer-to-character function

Syntax **chr (i : int) : char**

Description The **chr** function is used to convert an integer to a character. The character is the *i*-th character of the ASCII sequence of characters (except on the IBM mainframe, which uses the EBCDIC sequence.) For example, **chr** (65) is "A".

The **ord** function is the inverse of **chr**, so for any character *c*:

chr (**ord** (*c*)) = *c*.

See also **ord**, **intstr** and **strint** functions.

class declaration

Syntax A classDeclaration is:

```
[ monitor ]
class id
    [ inherit inheritItem ]
    [ implement implementItem ]
    [ implement by implementByItem ]
    [ import [ var ] importItem {, [ var ] importItem } ]
    [ export [ howExport ] id {, [ howExport ] id } ]
    statementsAndDeclarations
end id
```

Description A class declaration defines a template for a package of variables, constants, types, subprograms, etc. The name of the class (*id*) is given in two places, just after **class** and just after **end**. Items declared inside the class can be accessed outside of the class only if they are exported. Items from outside the class that are to be used in the class, need to be imported (unless they are predefined or pervasive). Instances (objects) of a class are created using the **new** statement. Each object is essentially a module located by a pointer.

Example This class is a template for creating objects, each of which is a stack of strings. (See the **module** description for the corresponding module that implements a single stack of strings.)

```
class stackClass    % Template for creating individual stacks
export push, pop

var top : int := 0
var contents : array 1 .. 100 of string

procedure push ( s : string )
    top := top + 1
    contents ( top ) := s
end push
```



```

procedure pop ( var s : string )
    s := contents ( top )
    top := top - 1
end pop
end stackClass

```

```

var p: pointer to stackClass % Short form: var p: ^stackClass
new stackClass, p           % Short form: new p

```

```

p -> push ( "Harvey" )
var name : string
    p -> pop ( name ) % This sets name to be Harvey

```

Pointer *p* is used to locate individual objects of the class. The **new** statement creates one of these objects. The statement

```
p -> push ( "Harvey" )
```

is a short form for:

```
stackClass (p) . push ("Harvey")
```

This inserts the string *Harvey* into the stack object located by *p*.

Details

The **new** statement is used to create objects of a class. Many instances of a class can exist at a given time, each located by a pointer. The **free** statement is used to destroy objects that are no longer of use. Turing does not support *garbage collection* (automatic recovery of space belonging to inaccessible objects).

See **modules** for a discussion of importing, exporting and related concepts. When an object is created by **new**, its initialization code is executed. In this example, the object's *top* variable is set to 0. As is true in modules, an exported subprogram of an object's class cannot be called until the object is completely initialized.

You are not allowed to create variables of a class, as in:

```
var s : stack           % Not legal!
```

If the **monitor** keyword is present (just before **class**), the objects are monitors. This means that only one process at a time can be active in the object. See **monitor** and **process**.

Inherit lists are used to specify inheritance. See **inherit** list. Implement and implement-by lists provide a special kind of expansion which supports the separation of an interface from its implementation. See **implement** list and **implement-by** list. A class cannot contain both an inherit and an implement list.

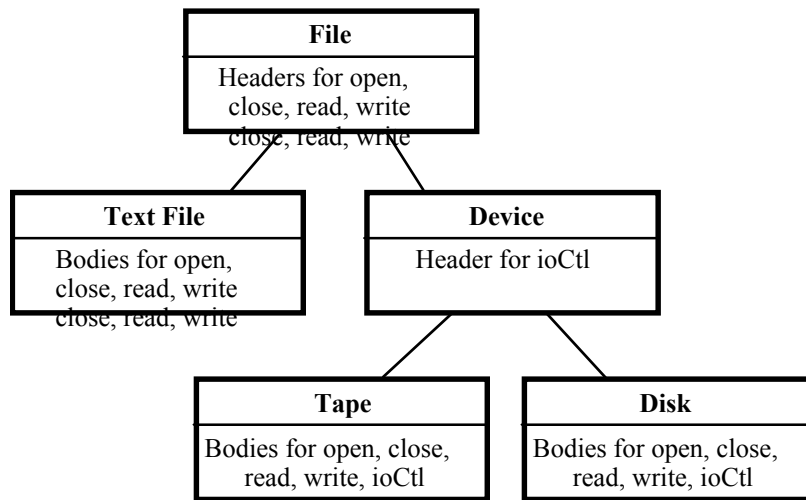
Class declarations can be nested inside modules and monitors but cannot be nested inside other classes or inside procedures or functions. A class must not contain a **bind** as one of its (outermost) declarations. A **return** statement cannot be used as one of the (outermost) statements in a class.

A class cannot export variables (or run time constants) as **unqualified** (because each object has a distinct set of variables).

The syntax of a *classDeclaration* presented above has been simplified by leaving out **pre**, **invariant** and **post** clauses. The full syntax which supports **pre**, **invariant** and **post** is the same as that for modules. The initialization of classes is the same as that for modules. See **module**.

Example

We will give an example in which a subprogram in one class overrides the corresponding subprogram in a class that is being inherited. The example is based on a program that implements a file system inside an operating system. All files have *open*, *close*, *read* and *write* operations. Some files, called *Device* files, also have an operation called *ioCtl* (input/output control). The kind of file determines the implementation method. Here is the expansion (inheritance) hierarchy among the classes of files.



The class called *File* gives the interface to all possible kinds of files. The *TextFile* class implements files that are text (ASCII characters). The *Device* class gives the interface to all files that have the *ioCtl* operation in addition to *open*, *close*, *read* and *write*. The *Tape* and *Disk* classes implement files that are actually physical tapes or disks. Here is the declaration of the *File* class:

```

class File
  export open, close, read, write
  deferred procedure open ( ... parameters for open ... )
  deferred procedure close ( ... parameters for close ... )
  deferred procedure read ( ... parameters for read ... )
  deferred procedure write ( ... parameters for write ... )
end File
  
```

The *TextFile* class implements the *File* interface by giving variables declarations and procedure bodies for ASCII files:

```
class TextFile
  inherit File
  var internalTextFileData :
    ... internal data for text files ...

  body procedure open
    ... body for open for text files ...
  end open
```

```
... bodies for close, read and write procedures for text files...
end TextFile
```

Objects to represent individual text files are created using the **new** statement:

```
var textFilePtr : ^ TextFile
    % Pointer will locate a text file object
new textFilePtr    % Create a text file object

    textFilePtr -> read ( ... actual parameters ... ) % Read text file
```

The *Device* class adds the *ioCtl* procedure to the *File* interface.

```
class Device
  inherit File
  export ioCtl
  deferred procedure ioCtl ( ... parameters for ioCtl ... )
end Device
```

The *Disk* class provides data and procedures to implement a file that is actually a disk (the *Tape* class is analogous):

```
class Disk
  inherit Device
  var internalDiskFileData : ... internal data for disk files

  body procedure open
    ... body for open ...
  end open

  ... bodies for close, read, write and ioCtl procedures for disk ...
end Disk
```

A pointer that can locate any kind of *File* object is declared this way:

```
var filePtr : ^ File
```

This may locate, for example, a *TextFile*:

```
filePtr := textFilePtr
```

This assignment is allowed because *filePtr*'s corresponding class (*File*) is an ancestor of *textFilePtr*'s corresponding class (*TextFile*). It is guaranteed that the object now located by *filePtr* supports a version of all the operations of a *File* (*open*, *close*, *read* and *write*).

When we call a procedure in the object located by *filePtr*, the actual procedure called will depend upon the object:

filePtr -> read (... actual parameters ...)

For example, if *filePtr* currently locates a *Disk* file, this will call the *read* procedure from the *Disk* class. This is an example of *dynamic binding* in which the version of *read* to be used is selected at run time and this choice is based on the object located by *filePtr*. This is called *polymorphism*, because *File* objects can have more than one form.

Example

As another example, consider class *C*, which contains headers and bodies for functions *f* and *g*. *C* exports functions *f* and *g*. There is also a class *D*, which inherits from *C*. Class *D* contains a body that overrides the body for *g*. *D* also contains a header and body for function *h*. *D* exports function *h*.

Pointer *p* has been declared to locate an object of class *C*, but at runtime *p* locates an object of class *D*. When *p* is used to call *f*, by means of *p*->*f*, the body of *f*, which appears in *C*, is invoked. When *p* is used to call *g*, by means of *p*->*g*, *g*'s overriding body in *D* is invoked. Any attempt to use *p* to call *h* is illegal because *p* can only be used to call functions that are exported from *C*.

```
class C
  export f, g

  procedure f
    put "C's f"
  end f

  procedure g
    put "C's g"
  end g
end C

class D
  inherit C           % Inherit f and g
  body procedure g    % Overrides g in C
    put "*** D's g ***"
  end g

  procedure h
    put "*** D's h ***"
  end h
end D

var p : pointer to C  % p can point to any descendant of C
new D, p              % p locates an object of class D
p -> f                 % Outputs "C's f"
p -> g                 % Outputs "*** D's g ***"
p -> h                 % Causes error "'h' is not in export list of 'C'"
```

See also **module**, **monitor** and **unit**. See also **import** list, **export** list, **implement** list, **implement by** list, and **inherit** list. See also **deferred** subprogram. See also **anyclass** and **objectclass**.

clock millisecs used procedure

Syntax	clock (var <i>c</i> : int)
Description	The clock statement is used to determine the amount of time since a program (process) started running. Variable <i>c</i> is assigned the number of milliseconds since the program started running.
Example	<p>This program tells you how much time it has used.</p> <pre>var timeRunning : int clock (timeRunning) put "This program has run ", timeRunning, " milliseconds"</pre>
Details	<p>On IBM PC compatibles, this is the total time since the Turing system was started up. The hardware resolution of duration is in units of 55 milliseconds. For example, clock(<i>i</i>) may be off by as much as 55 milliseconds.</p> <p>On Apple Macintoshes, this is the total time since the machine was turned on. The hardware resolution of duration is in units of 17 milliseconds (1/60-th of a second).</p>
See also	delay , time , sysclock , wallclock and date statements. See also predefined unit Time .

close file statement

Syntax	<p>A <i>closeStatement</i> is:</p> <p>close : fileNumber</p>
Description	In Turing, files are read and written using a <i>fileNumber</i> . In most cases, this number is given a value using the open statement, which translates a file name, such as "Master", to a file number, such as 5. When the program is finished using the file, it disconnects from the file using the close statement.
Example	This program illustrates how to open, read and then close a file.

```

var fileName : string := "Master" % Name of file
var fileNo : int % Number of file
var inputVariable : string ( 100 )
open : fileNo, fileName, read
...
read : fileNo, inputVariable
...
close : fileNo

```

Details	<p>In a Turing implementation, there will generally be a limit on the number of currently open files. This limit will typically be around 10. To avoid exceeding this limit, a program that uses many files one after another should close files that are no longer in use.</p> <p>If a program does not close a file, the file will be automatically closed when the program finishes.</p> <p>There is an older and still acceptable version of close that has this syntax:</p> <pre>close (fileNumber : int)</pre>
See also	the open , get , put , read , write , seek and tell statements.

cls clear screen graphics procedure

Syntax	cls
Description	The cls (clear screen) procedure is used to blank the screen. The cursor is set to the top left (to row 1, column 1).
Details	<p>In "<i>graphics</i>" mode all pixels are set to color number 0, so the screen is displayed in background color. In screen mode, the screen is set to the text background color.</p> <p>The screen should be in a "<i>screen</i>" or "<i>graphics</i>" mode. If the screen mode has not been set, it will automatically be set to "<i>screen</i>" mode. See setscreen for details.</p> <p>See also predefined unit Draw and Text.</p>

collection declaration

Syntax	<p>A <i>collectionDeclaration</i> is one of:</p> <ul style="list-style-type: none">(a) var <i>id</i> { , <i>id</i> } : collection of <i>typeSpec</i>(b) var <i>id</i> { , <i>id</i> } : collection of forward <i>typeId</i>
Description	<p>A collection declaration creates a new collection (or collections). A collection can be thought of as an array whose elements are dynamically created (by new) and deleted (by free). Elements of a collection are referred to by the collection's name subscripted by a pointer. See also new, free and pointer.</p>
Example	<p>Create a collection that will represent a binary tree.</p> <pre>var tree : collection of record name : string (10) left, right : pointer to tree end record var root : pointer to tree new tree, root tree (root) . name := "Adam"</pre>
Details	<p>The statement "new <i>C,p</i>" creates a new element in collection <i>C</i> and sets <i>p</i> to point at <i>i</i>. If there is no more memory space for the element, though, <i>p</i> is set to <i>nil</i> (<i>C</i>), which is the null pointer for collection <i>C</i>. The statement "free <i>C,p</i>" deletes the element of <i>C</i> pointed to by <i>p</i> and sets <i>p</i> to <i>nil</i> (<i>C</i>). In each case, <i>p</i> is passed as a var parameter and must be a variable of the pointer type of <i>C</i>.</p> <p>The keyword forward (form b above) is used to specify that the <i>typeId</i> of the collection elements will be given later in the collection's scope. The later declaration must appear at the same level (in the same list of declarations and statements) as the original declaration. This allows cyclic collections, for example, when a collection contains pointers to another collection, which in turn contains pointers to the first collection. In this case, the <i>typeId</i> is the name of the type that has not yet been declared; <i>typeId</i> cannot be used until its declaration appears. A collection whose element type is forward can be used only to declare pointers to it until the type's declaration is given.</p> <p>Suppose pointer <i>q</i> is equal to pointer <i>p</i> and the element they point to is deleted by "free <i>C,p</i>". We say <i>q</i> is a <i>dangling pointer</i> because it seems to locate an element, but the element no longer exists. A dangling pointer is considered to be an uninitialized value. It cannot be assigned, compared, used as a collection subscript, or passed to free.</p>

Collections cannot be assigned, compared, passed as parameters, bound to, or named by a **const** declaration. Collections must not be declared in procedures, functions, records or unions.

The same short forms for classes can be also used for collections. These include omission of the collection name in **new**, **free** and **nil** together with the **^** and **->** notations. Pointers to types (see **pointer**) can also be used, which are often more convenient to use than collections.

The syntax of a *collectionDeclaration* presented above has been simplified by leaving out **unchecked** collections. With this feature, a *collectionDeclaration* is one of:

(a) **var** *id* { , *id* } : [**unchecked**] **collection of** *typeSpec*

(b) **var** *id* { , *id* } : [**unchecked**] **collection of forward** *typeId*

When **unchecked** is specified, the checking to verify that pointers actually locate elements is removed. This checking is done using a "time stamp" attached to each element and pointer, and making sure that these match with each other. When **unchecked** is specified, the execution is dangerous, but faster and smaller, and the pointers become simply machine addresses (as in C).

color text color graphics procedure

Syntax **color** (*Color* : **int**)

Description The **color** procedure is used to change the currently active color. This is the color of characters that are to be **put** on the screen. The alternate spelling is **colour**.

Example This program prints out the message "Bravo" three times, each in a different color.

```
setscreen ( "graphics" )
for i : 1 .. 3
  color ( i )
  put "Bravo"
end for
```

Example This program prints out a message. The color of each letter is different from the preceding letter. For letter number *i* the color number is *i* mod maxcolor + 1. This cycles repeatedly through all the available colors.

```
setscreen ( "screen" )
const message := "Happy New Year!!"
```



```

for i : 1 .. length ( message )
  color ( i mod maxcolor + 1 )
  put message ( i ) ..
end for

```

- Details See **setscreen** for the number of colors available in the various "graphics" modes.
- The screen should be in a "screen" or "graphics" mode. If the screen mode has not been set, it will automatically be set to "screen" mode. See **setscreen** for details.
- See also **colorback**, **whatcolor**, **whattextcolor** and **maxcolor**.
- See also predefined unit **Text**.

colorback background color procedure

- Syntax **colorback** (*Color* : **int**)
- Description The **colorback** procedure is used to change the current background color. The alternate spelling is **colourback**.
- In "screen" mode on IBM PC compatibles, **colorback** sets the background color to one of the colors numbered 0 to 7. This is the color that surrounds characters when they are **put** onto the screen. On UNIX dumb terminals, **colorback**(1) turns on highlighting and **colorback**(0) turns it off. On other systems, this procedure may have no effect.
- In "graphics" mode on IBM PC compatibles, **colorback** is used to associate a color with pixel color number 0, which is considered to be the background color. Using **colorback** immediately changes the color being displayed for all pixels with color number 0.
- Example Since this program is in "screen" mode, changing the background color has no immediately observable effect. When the message "Greetings" is output, the background surrounding each letter will be in color number 2.
- ```

setscreen ("screen")
...
colorback (2)
 put "Greetings"

```
- Example            Since this program is in "graphics" mode, changing the background color immediately changes the colors of all pixels whose color number is 0.
- ```

setscreen ( "graphics" )
...

```

colorback (2)

Details	The screen should be in a "screen" or "graphics" mode. If the screen mode has not been set, it will automatically be set to "screen" mode. See setscreen for details
See also	color and whatcolorback . See also predefined unit Text .

comment remark statement

Description	A <i>comment</i> is a remark to the reader of the program, which the computer ignores. The most common form of comment in Turing starts with a percent sign (%) and continues to the end of the current line; this is called an <i>end-of-line</i> comment. There is also the <i>bracketed</i> comment, which begins with the /* and ends with */ and which can continue across line boundaries.
-------------	--

Example

```
% This is an end-of-line comment
var x : real           % Here is another end-of-line comment
const s := "Hello"
/* Here is a bracketed comment that
   lasts for two lines */
const pi := 3.14159
```

Details	In the BASIC language, comments are called <i>remarks</i> and start with the keyword REM. In Pascal, comments are bracketed by (* and *).
---------	---

comparisonOperator

Syntax	A <i>comparisonOperator</i> is one of:
--------	--

- (a) < % *Less than*
- (b) > % *Greater than*
- (c) = % *Equal*

- (d) `<=` % *Less than or equal; subset*
- (e) `>=` % *Greater than or equal; superset*
- (f) `not=` % *Not equal*

Description	<p>A comparison operator is placed between two values to determine their equality or ordering. For example, <code>7 > 2</code> is true and so is <code>"Adam" < "Cathy"</code>. The comparison operators can be applied to numbers as well as to enumerated types. They can also be applied to strings to determine the <i>ordering</i> between strings (see the string type for details). Arrays, records, unions and collections cannot be compared. Boolean values (true and false) can be compared only for equality (<code>=</code> and <code>not=</code>); the same is true of pointer values. Set values can be compared using <code><=</code> and <code>>=</code>, which are the subset and superset operators. The <code>not=</code> operator can be written as <code>~=</code>.</p> <p>Comparisons among classes is also supported (see class). If <i>C</i> and <i>D</i> are classes, <code>C <= D</code> means <i>D</i> is a descendant of (inherits from) <i>C</i>. See class.</p>
See also	<p>See also <i>infix</i> operators and <i>precedence</i> of operators. See also the int, real, string, set, boolean and enum types. See also string comparison.</p>

Concurrency All

Description	<p>This unit contains the predefined procedures that deal with concurrency. It contains one predefined function, although conceptually it contains three other subprograms.</p> <p>All routines in the Concurrency module are exported unqualified. (This means you can call the entry points directly.)</p>								
Entry Points	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="vertical-align: top; width: 20%;">empty*</td><td>Returns true if no processes are waiting on the condition queue.</td></tr> <tr> <td style="vertical-align: top;">getpriority*</td><td>Returns the priority of the current process.</td></tr> <tr> <td style="vertical-align: top;">setpriority*</td><td>Sets the priority of the current process.</td></tr> <tr> <td style="vertical-align: top;">simutime</td><td>Returns the number of simulated time units that have passed.</td></tr> </table> <p>* Part of the language, conceptually part of the Concurrency unit.</p>	empty *	Returns true if no processes are waiting on the condition queue.	getpriority *	Returns the priority of the current process.	setpriority *	Sets the priority of the current process.	simutime	Returns the number of simulated time units that have passed.
empty *	Returns true if no processes are waiting on the condition queue.								
getpriority *	Returns the priority of the current process.								
setpriority *	Sets the priority of the current process.								
simutime	Returns the number of simulated time units that have passed.								

Concurrency.empty

All

Syntax	empty (<i>variableReference</i>) : boolean
Description	The empty function is used in a concurrent program. It returns true if the <i>variableReference</i> , which must be a condition variable, has no processes waiting for it. Processes join the queue of a condition variable by executing the wait statement, and are awakened by the signal statement.
Status	Part of the language and only conceptually part of the Concurrency unit. This means that you can only call the function by calling empty , not by calling Concurrency.empty .
See also	condition , wait , signal , fork and monitor .

Concurrency.getpriority

All

Syntax	getpriority : nat
Description	The getpriority function returns the priority of an executing process in a concurrent program. A smaller value means a faster speed.
Status	Part of the language and only conceptually part of the Concurrency unit. This means that you can only call the function by calling getpriority , not by calling Concurrency.getpriority .
See also	setpriority , fork and monitor .

Concurrency.setpriority

All

Syntax	setpriority (<i>p</i> : nat)
Description	The setpriority procedure is used to set the priority of a process in a concurrent program. This priority cannot be counted on to guarantee critical access to shared variables. A smaller value of <i>p</i> means increased speed. The argument to setpriority may be limited to the range 0 to $2^{15} - 1$.
Status	Part of the language and only conceptually part of the Concurrency unit. This means that you can only call the function by calling setpriority , not by calling Concurrency.setpriority .
See also	getpriority , fork and monitor .

Concurrency.simutime

All

Syntax	simutime : int
Description	The simutime function returns the number of simulated time units that have passed since program execution began.
Details	Simulated time only passes when all process are either paused or waiting. This simulates the fact that CPU time is effectively infinitely faster than "pause" time.
Example	<p>This prints out the simulated time passing between two processes. This will print out 3, 5, 6, 9, 10, 12, 15, 15, 18, 20, 21, ...</p> <pre>process p (t : int) loop pause t put simutime end loop end p fork p (3)</pre>

fork *p* (5)

Status Exported unqualified.
This means that you can call the function by calling **simutime** or by calling **Concurrency.simutime**.

condition declaration

Syntax A conditionDeclaration is:

var *id* { , *id* } : [**array** *indexType* { , *indexType* } **of**]
 [*conditionOption*] **condition**

Description A condition is essentially a queue of sleeping processes. It is used in a concurrent program to allow processes to block themselves (by the **wait** statement) and later to be awakened (by the **signal** statement). A condition variable, which can occur only inside a monitor (a special kind of module that handles concurrency) or monitor class, is used by the **wait** and **signal** statements for putting processes to sleep and later waking them up.

Example The processes use this monitor to gain exclusive access to a resource. A process wanting to use the resource calls the *request* entry point and is blocked until the resource is free. When the process is finished with the resource, it calls the *release* entry point. This monitor is essentially a binary *semaphore* in which the semaphore's *P* operation is the *request* and the *V* is the *release*.

```
monitor resource
export request, release

var available : boolean := true
var nowAvailable : condition

procedure request
  if not available then
    wait nowAvailable % Go to sleep
  end if
  assert available
  available := false       % Allocate resource
end request

procedure release
  assert not available    % Resource is allocated
  available := true       % Free the resource
  signal nowAvailable    % Wake up one process
  % If any are sleeping
```

```

        end release

    end resource

    process worker
    loop
        ...
        resource.request      % Block until available
        ... use resource ...
        resource.release
    end loop
end worker

fork worker      % Activate one worker
fork worker      % Activate another worker

```

Details A *conditionOption* is one of:

- (a) **priority**
- (b) **deferred**
- (c) **timeout**

The **priority** option requires that the corresponding **wait** statements include priorities. Options (b) and (c) declare *deferred* conditions. A signal to a deferred condition causes the signaled process to become ready to enter the monitor when the monitor becomes inactive. The signaling process continues running in the monitor. A signal to an *immediate* (non deferred) condition causes the signaled process to begin running in the monitor immediately. The signaling process waits to re-enter the monitor when the monitor becomes inactive. All conditions in a device monitor must be deferred (or **timeout**).

A **timeout** option means the signaling is deferred and that an extra parameter to the **wait** statement must give a *timeout interval*. If a process waits longer than its interval, it is automatically signaled. Beware that the *empty* function can be non-repeatable when applied to timeout conditions. For example, **empty**(c) may not be equal to **empty**(c) in a single expression. In the current (1999) version of Turing, the time for time outs is measured in simulation time rather than real time. See the **pause** statement.

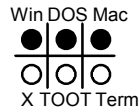
Conditions cannot be named as types, cannot be contained in records, unions or collections and cannot be declared in statements (such as **begin** or **loop**) or in subprograms. They can only be declared in monitors and monitor classes.

There is no guaranteed order of progress among awakened deferred processes, processes signaling immediate conditions, and processes attempting to enter an active monitor.

Note that *conditionOption* must precede the keyword **condition**.

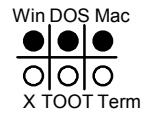
See also **wait** and **signal**. See also **monitor** and **fork**. See also **empty**. See also **pause**.

Config



Description	<p>This unit contains the predefined subprograms that deal with getting configuration information about the machine and environment on which the program is being run. It exists in order to allow users to obtain information about the system that may only be available at run time.</p> <p>All routines in the Config module are exported qualified (and thus must be prefaced with "Config.").</p>	
Entry Points	Display	Returns information about the display currently attached.
	Lang	Returns information about the language environment that the program is currently running within.
	Machine	Returns information about the hardware on which the program is running.

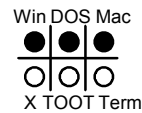
Config.Display



Syntax	Config.Display (<i>displayCode</i> : int) : int										
Description	<p>Config.Display returns information about the display (or displays) attached to the computer. The parameter <i>displayCode</i> determines what sort of information is passed back. <i>displayCode</i> has a number of possible values, all summarized by a set of predefined constants.</p> <p>At the time of this writing, the following constants were defined:</p> <table> <tr> <td><i>cdScreenHeight</i></td><td>return the height of the screen in pixels.</td></tr> <tr> <td><i>cdScreenWidth</i></td><td>return the width of the screen in pixels.</td></tr> <tr> <td><i>cdMaxNumColors</i></td><td>return the maximum number of colors supported by the display.</td></tr> <tr> <td><i>cdMaxNumColours</i></td><td>return the maximum number of colors supported by the display.</td></tr> <tr> <td><i>cdMaxNumPages</i></td><td>return the maximum number of display pages supported by the display (DOS OOT only).</td></tr> </table>	<i>cdScreenHeight</i>	return the height of the screen in pixels.	<i>cdScreenWidth</i>	return the width of the screen in pixels.	<i>cdMaxNumColors</i>	return the maximum number of colors supported by the display.	<i>cdMaxNumColours</i>	return the maximum number of colors supported by the display.	<i>cdMaxNumPages</i>	return the maximum number of display pages supported by the display (DOS OOT only).
<i>cdScreenHeight</i>	return the height of the screen in pixels.										
<i>cdScreenWidth</i>	return the width of the screen in pixels.										
<i>cdMaxNumColors</i>	return the maximum number of colors supported by the display.										
<i>cdMaxNumColours</i>	return the maximum number of colors supported by the display.										
<i>cdMaxNumPages</i>	return the maximum number of display pages supported by the display (DOS OOT only).										

Example	<p>This program prints the screen width and height.</p> <pre> const width : int := Config.Display (cdScreenWidth) const height: int := Config.Display (cdScreenHeight) put "The screen width is ", width, " the screen height is ", height </pre>
Details	<p>On the Macintosh, it's possible to have multiple displays attached to a single computer. To get information about the extra displays, you can call Config.Display with any of the first four constants above plus one, two, three, etc. This will return the height, width or maximum number of colors for the second, third and beyond displays.</p>
Example	<p>This program prints the screen width and height of the second display on a Macintosh.</p> <pre> const width : int := Config.Display (cdScreenWidth + 1) const height: int := Config.Display (cdScreenHeight + 1) put "The second display size is ", width, " x ", height </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Config.Display, not by calling Display.</p>
See Also	<p>View.SetActivePage, View.SetVisiblePage for details about multiple pages (<i>clMaxNumPages</i>).</p>

Config.Lang



Syntax	Config.Lang (<i>langCode</i> : int) : int				
Description	<p>Config.Lang returns information about the language and the limitations of the implementation that the program is currently running. The parameter <i>langCode</i> determines what sort of information is passed back. <i>langCode</i> has a number of possible values, all summarized by a set of predefined constants.</p> <p>At the time of this writing, the following constants were defined:</p> <table> <tr> <td><i>clRelease</i></td><td>return the current release number of the environment (e.g. 7.02 = 702).</td></tr> <tr> <td><i>clLanguageVersion</i></td><td>return the current version number of the language (e.g. 1.81 = 181).</td></tr> </table>	<i>clRelease</i>	return the current release number of the environment (e.g. 7.02 = 702).	<i>clLanguageVersion</i>	return the current version number of the language (e.g. 1.81 = 181).
<i>clRelease</i>	return the current release number of the environment (e.g. 7.02 = 702).				
<i>clLanguageVersion</i>	return the current version number of the language (e.g. 1.81 = 181).				

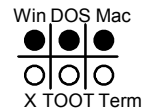
<i>clMaxNumStreams</i>	return the maximum number of I/O streams (used by the open and close statements) that can be opened at once.
<i>clMaxNumDirStreams</i>	return the maximum number of directory streams that can be opened at once.
<i>clMaxNumRunTimeArgs</i>	return the maximum number of run-time arguments.

Example This program prints the current environment version.

```
const version : int := Config.Lang (clLanguageVersion)
put "The language version number is ", version
```

Status Exported qualified.
This means that you can only call the function by calling **Config.Lang**, not by calling **Lang**.

Config.Machine



Syntax **Config.Machine** (*machineCode* : int) : int

Description **Config.Machine** returns information about the machine that the program is currently running on. The parameter *machineCode* determines what sort of information is passed back. *machineCode* has a number of possible values, all summarized by a set of predefined constants.

At the time of this writing, the following constants were defined:

<i>cmProcessor</i>	return an encoding of the processor number.
<i>cmFPU</i>	return 1 if there is an FPU installed, 0 if not.
<i>cmOS</i>	return the current version number of the operating system (e.g. 6.07 = 607).

Example This program prints whether the machine has an FPU or not.

```
if Config.Machine (cmFPU) = 1 then
  put "The machine has an FPU installed"
else
  put "The machine does not have an FPU installed"
end if
```

Status Exported qualified.
This means that you can only call the function by calling **Config.Machine**, not by calling **Machine**.

const constant declaration

Syntax A *constantDeclaration* is:

const *id* [: *typeSpec*] := *initializingValue*

Description A const declaration creates a name *id* for a value.

Example

```
const c := 3
const s := "Hello"       % The type of s is string
const x := sin ( y ) ** 2
const a : array 1..3 of int := init ( 1, 2, 3 )
const b : array 1..3 of int := a
const c : array 1..2, 1..2 of int := init ( 1, 2, 3, 4 )
                             % So c(1,1)=1, c(1,2)=2, c(2,1)=3, c(2,2)=4
```

Details The initializing value can be an arbitrary value or else a list of items separated by commas inside **init** (...). The syntax of *initializingValue* is:

- a. *expn*
- b. **init** (*initializingValue*, *initializingValue*)

Each **init** (...) corresponds to an array, record or union value that is being initialized. These must be nested for initialization of nested types. In the Pascal language, constants must have values known at compile time; Turing has no such restriction.

When the *typeSpec* is omitted, the variable's type is taken to be the (root) type of the initializing expression, for example, **int** or **string**. The *typeSpec* cannot be omitted for dynamic arrays or when the initializing value is of the form **init** (...). The values inside **init** (...) must be known at compile time.

The keyword **pervasive** can be inserted just after **const**. When this is done, the constant is visible inside all subconstructs of the constant's scope. Without **pervasive**, the constant is not visible inside modules, monitors or classes unless explicitly imported. Pervasive constants need not be imported. You can abbreviate **pervasive** as an asterisk (*).

You can also optionally use the **register** keyword to request that the constant be placed in a machine register. The syntax for *constantDeclaration* is actually:

const [**pervasive**] [**register**] *id* [: *typeSpec*] := *initializingValue*

In the current (1999) implementation, programs are run interpretively using pseudo-code, which has no machine registers, and the **register** keyword is ignored. See also **register** for restrictions on the use of register constants.

constantReference use of a constant

Syntax	<p>A <i>constantReference</i> is:</p> $\textit{constantId} \{ \textit{componentSelector} \}$
Description	<p>In a Turing program, a constant is declared and given a name (<i>constantId</i>) and then used. Each use is called a <i>constant reference</i>.</p> <p>If the constant is an array, record or union, its parts (<i>components</i>) can be selected using subscripts and field names (using <i>componentSelectors</i>). The form of a <i>componentSelector</i> is one of:</p> <p>(a) (<i>expn</i> {, <i>expn</i>})</p> <p>(b) . <i>fieldId</i></p> <p>Form (a) is used for subscripting (indexing) arrays. The number of array subscripts must be the same as in the array's declaration. Form (b) is used for selecting a field of a record or union. Component selectors are used in the same manner as variable references. See <i>variableReference</i> for details. See also const declaration and <i>explicitConstant</i>.</p>
Example	<pre>var radius : real const pi := 3.14159 % Constant declaration ... put "Area is: ", pi * radius **2</pre> <p style="text-align: right;">% pi is a constant reference.</p>

COS cosine function (radians)

Syntax	cos (<i>r</i> : real) : real
Description	<p>The cos function is used to find the cosine of an angle given in radians. For example, cos (0) is 1.</p>
Example	<p>This program prints out the cosine of $\pi/6$, $2\pi/6$, $3\pi/6$, up to $12\pi/6$ radians.</p> <pre>const pi := 3.14159 for i : 1 .. 12 const angle := i * pi / 6</pre>

```

put "Cos of ", angle, " is ", cos ( angle )
end for

```

See also **cosd** function which finds the cosine of an angle given in degrees. (2π radians are the same as 360 degrees.)
See also predefined unit **Math**.

cosd cosine function (degrees)

Syntax **cosd** (*r* : **real**) : **real**

Description The **cosd** function is used to find the cosine of an angle given in degrees. For example, **cosd** (0) is 1.

Example This program prints out the cosine of 30, 60, 90, up to 360 degrees.

```

for i : 1 .. 12
  const angle := i * 30
  put "Cos of ", angle, " is ", cosd ( angle )
end for

```

See also **cos** function which finds the cosine of an angle given in radians. (2π radians are the same as 360 degrees.)
See also predefined unit **Math**.

date procedure

Syntax **date** (**var** *d* : **string**)

Description The **date** statement is used to determine the current date. Variable *d* is assigned a string in the format "*dd mmm yy*", where *mmm* is the first 3 characters of the month, e.g., "*Apr*". For example, if the date is Christmas 1989, *d* will be set to "*25 Dec 89*".

Example This program greets you and tells you the date.

	<pre> var today : string date (today) put "Greetings!! The date today is ", today </pre>
Details	Be warned that on some computers, such as IBM PC compatibles or Apple Macintoshes, the date may not be set correctly in the operating system; in that case, the date procedure will give incorrect results.
See also	delay , clock , sysclock , wallclock and time statements. See also predefined unit Time .

declaration create a variable

Syntax	<p>A <i>declaration</i> is one of:</p> <ul style="list-style-type: none"> (a) <i>variableDeclaration</i> (b) <i>constantDeclaration</i> (c) <i>typeDeclaration</i> (d) <i>bindDeclaration</i> (e) <i>procedureDeclaration</i> (f) <i>functionDeclaration</i> (g) <i>moduleDeclaration</i> (h) <i>classDeclaration</i> (i) <i>processDeclaration</i> (j) <i>monitorDeclaration</i> (k) <i>conditionDeclaration</i>
Description	A <i>declaration</i> creates a new name (or names) for a variable, constant, type, procedure, function, module, class, process, monitor, or condition. These names are called <i>identifiers</i> , where <i>id</i> is the abbreviation for <i>identifier</i> .
Example	<pre> var width : int % Variable declaration const pi := 3.14159 % Constant declaration type range : 0 .. 150 % Type declaration procedure greet % Procedure declaration put "Hello world" end greet </pre>

Details	<p>Ordinarily, each new name must be distinct from names that are already visible; that is, redeclaration is not allowed. There are certain exceptions to this rule, for example, names of parameters and fields of records can be the same as existing visible variables. Variables declared inside a subprogram (a procedure and function) are allowed to be the same as variables global to (outside of) the subprogram.</p> <p>The effect of a declaration (its <i>scope</i>) lasts to the end of the construct in which the declaration occurs; this will be the end of the program, the end of the surrounding procedure, function or module, the end of a loop, for, case or begin statement, or the end of the then, elsif, or else clause of an if statement, or the end of the case statement alternative.</p> <p>A name must be declared before it can be used; this is called the <i>DBU</i> (<i>Declaration Before Use</i>) rule. The exceptions to this rule use the keyword forward, as in import lists and in collection declarations.</p> <p>A <i>declaration</i> can appear any place a <i>statement</i> can appear. This differs from the Pascal language, in which declarations are allowed only at the beginning of the program or at the beginning of a procedure or function. Each declaration can optionally be followed by a semicolon (;).</p> <p>There are certain restrictions on the placement of declarations. Procedures and functions cannot be declared inside other procedures and functions nor inside statements (for example, not inside an if statement). A bind declaration cannot appear at the outer level of either the main program or a module. A condition declaration can appear only inside a monitor. Processes cannot be declared inside procedures, functions, monitors or classes. Classes cannot be declared inside classes. However, modules and monitors can be declared inside classes and vice versa. Monitors can be declared inside modules, not vice versa.</p>
---------	---

deferred subprogram declaration

Syntax	A deferredDeclaration is: <div style="text-align: center;">deferred subprogramHeader</div>
Description	A procedure or function is declared to be deferred when you want to be able to override the subprogram in an expansion. The procedure or function must be in a module, monitor or class.
Example	The <i>display</i> procedure is deferred in this class of stacks to allow various ways of graphically displaying the stack on the screen: <pre>class stack</pre>

```

export push, pop
... local declarations ...
... declarations of the push and pop procedures ...
deferred procedure display ( howbig : int )
    end stack

```

An expansion to the *stack* class can give a body for *display*, as in:

```

class stackWithSimpleDisplay
    body procedure display          % ( howbig : int )
        ... graphically display the stack on the screen ...
    end display
end stackWithSimpleDisplay

```

The following creates a stack that can be displayed and displays it:

```

var p : ^stackWithSimpleDisplay
new p
...
    p -> display ( 25 )          % Display the stack on the screen

```

Details	<p>A deferred procedure is <i>resolved</i> by giving its body. This can be done in the scope (module, monitor or class) containing the deferred declaration (following the deferred declaration) or in any expansion of that scope. Only one resolution per scope is allowed. Unresolved subprograms can be called, but they immediately abort.</p> <p>All exported subprograms are implicitly deferred and can be overridden in expansions.</p> <p>During initialization of a module, monitor or object of a class, deferred subprograms (including exported subprograms) cannot be called. This restriction prevents accessing an object before it is fully initialized.</p> <p>A deferred declaration must not appear in the main program.</p>
See also	module , monitor and class . See also export list, import list, inherit list, implement list and implement by list.

delay procedure

Syntax	delay (<i>duration : int</i>)
Description	The delay statement is used to cause the program to pause for a given time. The time duration is in milliseconds.
Example	<p>This program prints the integers 1 to 10 with a second delay between each.</p> <pre>for <i>i</i> : 1 .. 10</pre>


```

put i
delay ( 1000 )% Pause for 1 second
end for

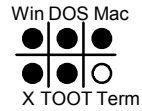
```

- Details** On IBM PC compatibles, the hardware resolution of duration is in units of 55 milliseconds. For example, **delay**(500) will delay the program by about half a second, but may be off by as much as 55 milliseconds.
- On Apple Macintoshes, the hardware resolution of duration is in units of 17 milliseconds (1/60th of a second). For example, **delay**(500) will delay the program by about half a second, but may be off by as much as 17 milliseconds.
- See also** **sound, clock, sysclock, wallclock, time** and **date** statements.
- See also predefined unit **Time**.



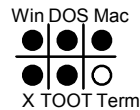
- Description** This unit contains the predefined subprograms that deal with directories. You can use these subprograms to list the contents of directories, create directories, change directories and return the current directory.
- All routines in the Dir module are exported qualified (and thus must be prefaced with "**Dir.**").
- Entry Points** **Open** Opens a directory stream in order to get a listing of the directory contents.
- Get** Gets the next file name in the directory listing.
- GetLong** Gets the next file name and other information in the directory listing.
- Close** Closes the directory stream.
- Create** Creates a new directory.
- Delete** Deletes a directory (must be empty).
- Change** Changes the current execution directory.
- Current** Returns the current execution directory.
- See also** **Files** unit for an explanation of the different ways of specifying a path name of a file or directory under the different operating systems.

Dir.Change



Syntax	Dir.Change (<i>directoryPathName</i> : string)
Description	Dir.Change changes the execution directory to that specified by the parameter <i>directoryPathName</i> . This is the equivalent of doing a cd in UNIX. On a Windows or DOS machine, specifying a drive in the <i>directoryPathName</i> parameter causes the drive to become the default drive (unlike the DOS cd command).
Details	If the Dir.Change call fails, then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.
Example	<p>This program changes to the directory called <i>/usr/west</i> and then lists the current directory.</p> <pre>Dir.Change ("/usr/west") if Error.Last = eNoError then put "Directory changed" else put "Did not change the directory." put "Error: ", Error.LastMsg end if put "The current execution directory is ", Dir.Current</pre>
Status	Exported qualified. This means that you can only call the function by calling Dir.Change , not by calling Change .

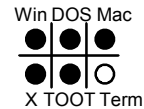
Dir.Close



Syntax	Dir.Close (<i>streamNumber</i> : int)
--------	---

Description	Dir.Close is part of a series of four subprograms that help users get directory listings. Dir.Close is used to close a directory stream number opened by Dir.Open . After the directory stream number is closed, it can not be used with Dir.Get or Dir.GetLong .
Details	If the Dir.Close call fails, then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.
Example	<p>This program prints a listing of all the files in the directory <i>datafiles</i>.</p> <pre> var streamNumber : int var fileName : string streamNumber := Dir.Open ("datafiles") assert streamNumber > 0 loop fileName := Dir.Get (streamNumber) exit when fileName = "" put fileName end loop Dir.Close (streamNumber) </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Dir.Close, not by calling Close.</p>

Dir.Create



Syntax	Dir.Create (<i>directoryPathName</i> : string)
Description	Dir.Create is used to create the directory specified by the parameter <i>directoryPathName</i> . This is the equivalent of doing a mkdir in DOS or UNIX. On the Macintosh, it creates a folder.
Details	If the Dir.Create call fails, then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.
Example	<p>This program creates the directory called <i>information</i>.</p> <pre> Dir.Create ("information") if Error.Last = eNoError then put "Directory created" else put "Did not create the directory." put "Error: ", Error.LastMsg </pre>

end if

Status Exported qualified.
This means that you can only call the function by calling **Dir.Create**, not by calling **Create**.

Dir.Current

Win DOS Mac
● ● ●
● ● ○
X TOOT Term

Syntax **Dir.Current** : **string**

Description **Dir.Current** returns the full path name of the current execution directory.
This is the equivalent of doing a **pwd** in UNIX.

Details If the **Dir.Current** call fails, then **Error.Last** will return a non-zero value
indicating the reason for the failure. **Error.LastMsg** will return a string
which contains the textual version of the error.

Example This program changes to the directory called */usr/west* and then lists the
current directory.

```
Dir.Change ("/usr/west")  
if Error.Last = eNoError then  
  put "Directory changed"  
else  
  put "Did not change the directory."  
  put "Error: ", Error.LastMsg  
end if  
  
  put "The current execution directory is ", Dir.Current
```

Status Exported qualified.
This means that you can only call the function by calling **Dir.Current**, not
by calling **Current**.

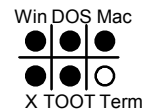
Dir.Delete

Win DOS Mac
● ● ●
● ● ○
X TOOT Term

Syntax **Dir.Delete** (*directoryPathName* : **string**)

Description	<p>Dir.Delete is used to delete the directory specified by the parameter <i>directoryPathName</i>. This is the equivalent of doing a rmdir in DOS or UNIX. On the Macintosh, it removes a folder.</p> <p>Dir.Delete will fail if it attempts delete a directory that has files in it.</p>
Details	<p>If the Dir.Delete call fails, then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.</p>
Example	<p>This program deletes the directory called <i>information</i>.</p> <pre> Dir.Delete ("information") if Error.Last = eNoError then put "Directory delete" else put "Did not delete the directory." put "Error: ", Error.LastMsg end if </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Dir.Delete, not by calling Delete.</p>

Dir.Get



Syntax	Dir.Get (<i>streamNumber</i> : int) : string
Description	<p>Dir.Get is part of a series of four subprograms that help users get directory listings. Dir.Get is used to get the file names in the directory. Each time the function is called, it returns the next file name in the directory. The names are not sorted. When there are no more file names in the directory, Dir.Get returns the empty string.</p>
Details	<p>If the Dir.Get call fails, then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.</p>
Example	<p>This program prints a listing of all the files in the directory <i>datafiles</i>.</p> <pre> var streamNumber : int var fileName : string streamNumber := Dir.Open ("datafiles") assert streamNumber > 0 loop fileName := Dir.Get (streamNumber) </pre>

Status Exported qualified.

This means that you can only call the function by calling **Dir.Get**, not by calling **Get**.



Description
<p>Dir.GetLong is part of a series of four subprograms that help users get directory listings. Dir.GetLong is used to get the names and assorted information of the files in the directory. Each time the function is called, it returns the name and information of the next file in the directory. The names are not sorted. When there are no more file names in the directory, Dir.GetLong returns the empty string in the <i>entryName</i> parameter.</p> <p>The <i>size</i> parameter is the size of the file in bytes. The <i>attribute</i> parameter has its individual bits set as follows (the individual bits can be extracted using the bits operator):</p> <ul style="list-style-type: none"> Bit 0 – <i>ootAttrDir</i> – set to 1 if entry is a directory. Bit 1 – <i>ootAttrRead</i> – set to 1 if the program can read the file. Bit 2 – <i>ootAttrWrite</i> – set to 1 if the program can write the file. Bit 3 – <i>ootAttrExecute</i> – set to 1 if the program can execute the file. Bit 4 – <i>ootAttrHidden</i> – set to 1 if the entry if a hidden file (PC, Mac). Bit 5 – <i>ootAttrSystem</i> – set to 1 if the entry is a system file (PC only). Bit 6 – <i>ootAttrVolume</i> – set to 1 if the entry is a volume name (PC only). Bit 7 – <i>ootAttrArchive</i> – set to 1 if the entry has archive bit set (PC only). Bit 8 – <i>ootAttrOwnerRead</i> – set to 1 if the file owner can read it. Bit 9 – <i>ootAttrOwnerWrite</i> – set to 1 if the file owner can write it. Bit 10 – <i>ootAttrOwnerExecute</i> – set to 1 if the file owner can execute it. Bit 11 – <i>ootAttrGroupRead</i> – set to 1 if the file owner's group members can read it. Bit 12 – <i>ootAttrGroupWrite</i> – set to 1 if the file owner's group members can write it. Bit 13 – <i>ootAttrGroupExecute</i> – set to 1 if the file owner's group members can execute it. Bit 14 – <i>ootAttrGroupRead</i> – set to 1 if anyone can read it. Bit 15 – <i>ootAttrGroupWrite</i> – set to 1 if anyone can write it.

Bit 16 – *ootAttrGroupExecute* – set to 1 if anyone can execute it.

Bits 8-16 are only really applicable to multiple user systems like UNIX, although they are set properly on any OOT system.

The *ootAttr...* constants are defined in the **File** unit. They correspond to the values of *attribute* if a specified bit is set. For example, *ootAttrSystem* is the value of the *attribute* parameter if bit 5 is set to 1. You can **and** or **or** these constants to get combinations of specific file attributes.

The *fileTime* is the time of last modification of the file. It is returned as the number of seconds since 00:00:00 GMT 1/1/1970. To convert this to a string, use **Time.SecDate**

Details If the **Dir.GetLong** call fails, then **Error.Last** will return a non-zero value indicating the reason for the failure. **Error.LastMsg** will return a string which contains the textual version of the error.

Example This program prints a listing of all the files in the directory *datafiles*.

```
var streamNumber : int
var fileName : string
var size, attribute, fileTime : int
streamNumber := Dir.Open ("datafiles")
assert streamNumber > 0
loop
  Dir.GetLong (streamNumber, fileName, size, attribute, fileTime)
  exit when fileName = ""
  put fileName, " ", Time.SecDate (fileTime)
end loop
Dir.Close (streamNumber)
```

Example This program prints a listing of the attributes of all the files in the current directory.

```
var streamNumber : int
var fileName : string
var size, attribute, fileTime : int
streamNumber := Dir.Open (Dir.Current)
assert streamNumber > 0
loop
  Dir.GetLong (streamNumber, fileName, size, attribute, fileTime)
  exit when fileName = ""
  put fileName, " ..
  if (attribute and ootAttrDir) not= 0 then
    put "Directory " ..
  end if
  if (attribute and ootAttrRead) not= 0 then
    put "Readable " ..
  end if
  if (attribute and ootAttrWrite) not= 0 then
    put "Writable " ..
  end if
  if (attribute and ootAttrExecute) not= 0 then
    put "Executable " ..
  end if
```

```

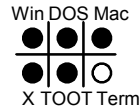
    put ""
end loop

```

Dir.Close (*streamNumber*)

Status Exported qualified.
This means that you can only call the function by calling **Dir.GetLong**, not by calling **GetLong**.

Dir.Open



Syntax **Dir.Open** (*directoryPathName* : **string**) : **int**

Description **Dir.Open** is part of a series of four subprograms that help users get directory listings. **Dir.Open** returns a directory stream number if the directory could be opened. This stream number can be used to get file names and information using the **Dir.Get** and **Dir.GetLong** subprograms. After getting the listing, the user should call **Dir.Close**.

Details If the **Dir.Open** call fails, then **Error.Last** will return a non-zero value indicating the reason for the failure. **Error.LastMsg** will return a string which contains the textual version of the error.

Example This program prints a listing of all the files in the current directory.

```

var streamNumber : int
var fileName : string
streamNumber := Dir.Open (Dir.Current)
assert streamNumber > 0
loop
  fileName := Dir.Get (streamNumber)
  exit when fileName = ""
  put fileName
end loop
Dir.Close (streamNumber)

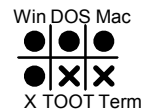
```

Status Exported qualified.
This means that you can only call the function by calling **Dir.Open**, not by calling **Open**.

div integer truncating division operator

Syntax	div
Description	The div operator divides one number by another and produces the integer result, truncated in the direction of zero. For example, <code>7 div 2</code> produces 3 and <code>-7 div 2</code> produces -3.
Example	<p>In this example, <code>eggCount</code> is the total number of eggs. The first put statement determines how many dozen eggs there are. The second put statement determines how many extra eggs there are beyond the last dozen.</p> <pre>var eggCount : int get eggCount put "You have ", eggCount div 12, " dozen eggs" put "You have ", eggCount mod 12, " left over"</pre>
See also	<i>infix operators, precedence of operators and the mod operator.</i>

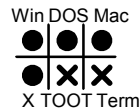
Draw



Description	<p>This unit contains the predefined subprograms that deal with drawing pixel graphics to the screen.</p> <p>All routines in the Draw unit are exported qualified (and thus must be prefaced with "Draw.").</p>	
Entry Points	Cls	Clears the screen to color 0.
	Dot	Draws a dot.
	Line	Draws a line.
	Box	Draws a box.
	FillBox	Draws a filled box.
	Oval	Draws an oval.

FillOval	Draws a filled oval.
Arc	Draws an arc.
FillArc	Draws a filled arc or a wedge.
Polygon	Draws a polygon.
FillPolygon	Draws a filled polygon.
MapleLeaf	Draws a maple leaf.
FillMapleLeaf	Draws a filled maple leaf.
Star	Draws a star.
FillStar	Draws a filled star.
Fill	Does a flood fill.
Text	Draws text as graphics

Draw.Arc



Syntax	Draw.Arc (<i>x</i> , <i>y</i> , <i>xRadius</i> , <i>yRadius</i> : int , <i>initialAngle</i> , <i>finalAngle</i> , <i>Color</i> : int)
Description	The Draw.Arc procedure is used to draw an arc whose center is at (<i>x</i> , <i>y</i>). This is just like Draw.Oval , except that you must also give two angles, <i>initialAngle</i> and <i>finalAngle</i> , which determine where to start and stop drawing. Zero degrees is "three o'clock", 90 degrees is "twelve o'clock", etc. The horizontal and vertical distances from the center to the arc are given by <i>xRadius</i> and <i>yRadius</i> .




Example	This program draws a semicircle (actually, an approximation to a semicircle) whose center is (<i>midx</i> ,0) the bottom center of the screen, using color number 1. The maxx and maxy functions are used to determine the maximum x and y values on the screen.
---------	---

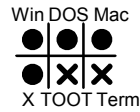
```
View.Set ( "graphics" )
const midx := maxx div 2
      Draw.Arc ( midx, 0, maxx, maxy, 0, 180, 1 )
```

Details	The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.
Status	Exported qualified. This means that you can only call the function by calling Draw.Arc , not by calling Arc .
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.Box

Syntax	Draw.Box (<i>x1</i> , <i>y1</i> , <i>x2</i> , <i>y2</i> , <i>Color</i> : int)
Description	The Draw.Box procedure is used to draw a box on the screen with bottom left and top right corners of (<i>x1</i> , <i>y1</i>) to (<i>x2</i> , <i>y2</i>) using the specified <i>Color</i> . 
Example	This program draws a large box, reaching to each corner of the screen using color number 1. The maxx and maxy functions are used to determine the maximum x and y values on the screen. The point (0,0) is the left bottom of the screen and (maxx , maxy) is the right top. View.Set ("graphics") Draw.Box (0, 0, maxx , maxy , 1)
Details	The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.
Status	Exported qualified. This means that you can only call the function by calling Draw.Box , not by calling Box .
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.Cls

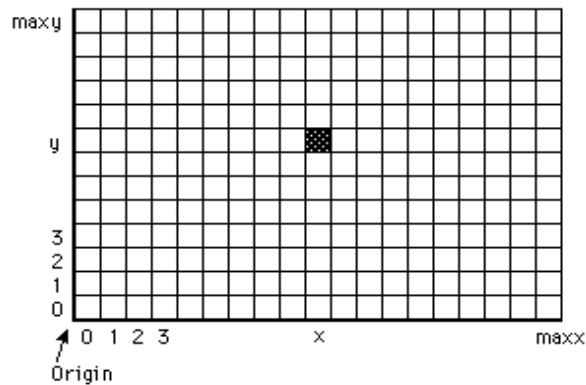


Syntax	Draw.Cls
Description	The Draw.Cls (clear screen) procedure is used to blank the screen. The cursor is set to the top left (to row 1, column 1).
Details	<p>In "<i>graphics</i>" mode all pixels are set to color number 0, so the screen is displayed in background color. In screen mode, the screen is set to the text background color.</p> <p>The screen should be in a "<i>screen</i>" or "<i>graphics</i>" mode. If the screen mode has not been set, it will automatically be set to "<i>screen</i>" mode. See View.Set for details.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Draw.Cls, not by calling Cls.</p>
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.Dot



Syntax	Draw.Dot (<i>x</i> , <i>y</i> , <i>Color</i> : int)
Description	The Draw.Dot procedure is used to color the dot (pixel) at location (<i>x</i> , <i>y</i>) using the specified <i>Color</i> .



Example This program randomly draws dots with random colors. The **maxx**, **maxy** and **maxcolor** functions give the maximum x, y and color values.

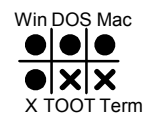
```
View.Set ( "graphics" )
var x, y, c : int
loop
  x := Rand.Int (0, maxx)      % Random x
  y := Rand.Int (0, maxy)      % Random y
  clr := Rand.Int (0, maxcolor) % Random color
  Draw.Dot ( x, y, c )
end loop
```

Details The screen should be in a "graphics" mode. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode. See **View.Set** for details.

Status Exported qualified.
This means that you can only call the function by calling **Draw.Dot**, not by calling **Dot**.

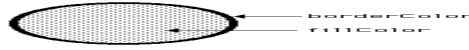
See also **View.Set**, **maxx**, **maxy** and the various procedures in the **Draw** unit.

Draw.Fill



Syntax **Draw.Fill** (x, y : **int**, fillColor, borderColor : **int**)

Description The **Draw.Fill** procedure is used to color in a figure that is on the screen. Starting at (x, y) , the figure is filled with *fillColor* to a surrounding border whose color is *borderColor*.



Example This program draws an oval with x and y radius of 10 in the center of the screen using color 1. Then the oval is filled with color 2. The **maxx** and **maxy** functions are used to determine the maximum x and y values on the screen.

```
View.Set ( "graphics" )
const midx := maxx div 2
const midy := maxy div 2
drawoval ( midx, midy, 10, 10, 1 )
      Draw.Fill ( midx, midy, 2, 1 )
```

Details The screen should be in a "graphics" mode. See the **View.Set** procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.

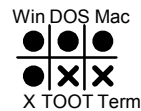
Warning: In Turing for IBM PC compatibles, **Draw.Fill** fails to completely fill in some complicated figures that contain "islands" within them surrounded by the *borderColor*.

Status Exported qualified.

This means that you can only call the function by calling **Draw.Fill**, not by calling **Fill**.

See also **View.Set**, **maxx**, **maxy** and the various procedures in the **Draw** unit.

Draw.FillArc



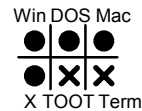
Syntax **Draw.FillArc** ($x, y, xRadius, yRadius$: **int**,
initialAngle, finalAngle, Color : **int**)

Description The **Draw.FillArc** procedure is used to draw a filled arc whose center is at (x, y) . It then fills in the pie-shaped wedge using the specified *Color*. To outline a filled arc, use **Draw.FillArc** with the *Color* parameter set to the fill color and then **Draw.Arc** with the *Color* parameter set to the border color. For *initialAngle* and *finalAngle*, which determine the edges of the wedge, zero degrees is "three o'clock" and 90 degrees is "twelve o'clock", etc. The horizontal and vertical distances from the center to the arc are given by *xRadius* and *yRadius*.

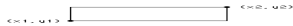


Example	<p>This program draws a filled semicircle (actually, an approximation to a semicircle) whose center is (<i>midx</i>,0) the bottom center of the screen, using color number 1. The maxx and maxy functions are used to determine the maximum x and y values on the screen.</p> <pre> View.Set ("graphics") const <i>midx</i> := maxx div 2 Draw.FillArc (<i>midx</i>, 0, maxy, maxy, 0, 180, 1) </pre>
Details	<p>On the PC, Draw.FillArc fills the pie-shaped wedge by using a "flood" fill and is thus subject to all the conditions of a flood fill.</p> <p>The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Draw.FillArc, not by calling FillArc.</p>
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.FillBox



Syntax	Draw.FillBox (<i>x1</i> , <i>y1</i> , <i>x2</i> , <i>y2</i> , <i>Color</i> : int)
Description	<p>The Draw.FillBox procedure is used to draw a filled box on the screen with bottom left and top right corners of (<i>x1</i>, <i>y1</i>) to (<i>x2</i>, <i>y2</i>) filled using the specified <i>Color</i>. To get a box outlined in a different color, use Draw.FillBox with the <i>Color</i> parameter set to the fill color and then call Draw.Box with the <i>Color</i> parameter set to the border color.</p>



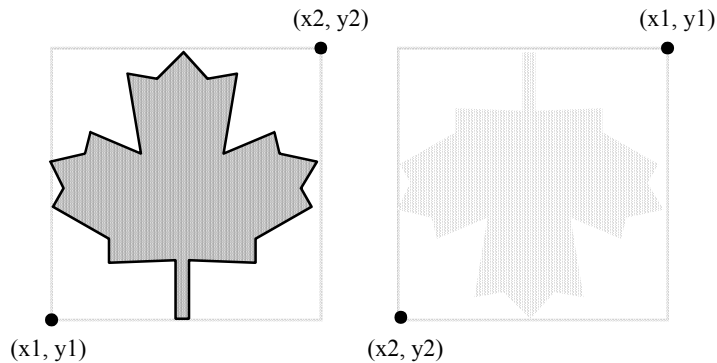
Example	<p>This program will fill the bottom half of the screen with color 1 and then outline it in color 2. The maxx and maxy functions are used to determine the maximum x and y values on the screen. The point (0,0) is the left bottom of the screen and (maxx, maxy) is the right top.</p> <pre> View.Set ("graphics") Draw.FillBox (0, 0, maxx, maxy div 2, 1) Draw.Box (0, 0, maxx, maxy div 2, 2) </pre>
---------	--

Details	The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.
Status	Exported qualified. This means that you can only call the function by calling Draw.FillBox , not by calling FillBox .
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.FillMapleLeaf



Syntax	Draw.FillMapleLeaf (<i>x1</i> , <i>y1</i> , <i>x2</i> , <i>y2</i> , <i>Color</i> : int)
Description	The Draw.FillMapleLeaf procedure is used to draw a filled maple leaf on the screen bounded by a rectangle with bottom left and top right corners of (<i>x1</i> , <i>y1</i>) to (<i>x2</i> , <i>y2</i>) and filled using the specified <i>Color</i> . To get a maple leaf outlined in a different color, use Draw.FillMapleLeaf with the <i>Color</i> parameter set to the fill color and then call Draw.MapleLeaf with the <i>Color</i> parameter set to the border color. If <i>y1</i> is greater than <i>y2</i> , then the mapleleaf is drawn upside down.



Example	This program will draw two maple leaves beside each other. The first will be outlined in color 1 and filled in color 2. The second maple leaf will be upside down and both filled and outlined in color 3.
---------	--

```

View.Set ( "graphics" )
Draw.FillMapleLeaf ( 0, 0, 100, 100, 1 )
Draw.MapleLeaf ( 0, 0, 100, 100, 2 )
Draw.FillMapleLeaf ( 150, 100, 250, 0, 3 )

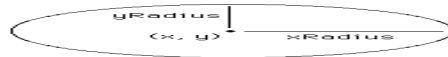
```


Details	<p>The Draw.FillMapleLeaf procedure is useful for drawing the Canadian flag.</p> <p>The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Draw.FillMapleLeaf, not by calling FillMapleLeaf.</p>
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.FillOval



Syntax	Draw.FillOval (<i>x</i> , <i>y</i> , <i>xRadius</i> , <i>yRadius</i> , <i>Color</i> : int)
Description	<p>The Draw.FillOval procedure is used to draw a filled oval whose center is at (<i>x</i>, <i>y</i>). The horizontal and vertical distances from the center to the oval are given by <i>xRadius</i> and <i>yRadius</i>. To get an oval outlined in a different color, use Draw.FillOval with the <i>Color</i> parameter set to the fill color and then call Draw.Oval with the <i>Color</i> parameter set to the border color.</p>



Example	<p>This program draws a large filled oval that just touches each edge of the screen using color number 1. The maxx and maxy functions are used to determine the maximum x and y values on the screen. The center of the oval is at (<i>midx</i>, <i>midy</i>), which is the middle of the screen.</p>
---------	---

```

View.Set ( "graphics" )
const midx := maxx div 2
const midy := maxy div 2
Draw.FillOval ( midx, midy, midx, midy, 1 )

```

Details	<p>Ideally, a circle is drawn when <i>xRadius</i> = <i>yRadius</i>. In fact, the aspect ratio (the ratio of height to width of pixels displayed on the screen) of the IBM PC compatibles is not 1.0, so this does not draw a true circle. In CGA graphics mode this ratio is 5 to 4.</p> <p>The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p>
---------	---

Status	Exported qualified. This means that you can only call the function by calling Draw.FillOval , not by calling FillOval .
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.FillPolygon



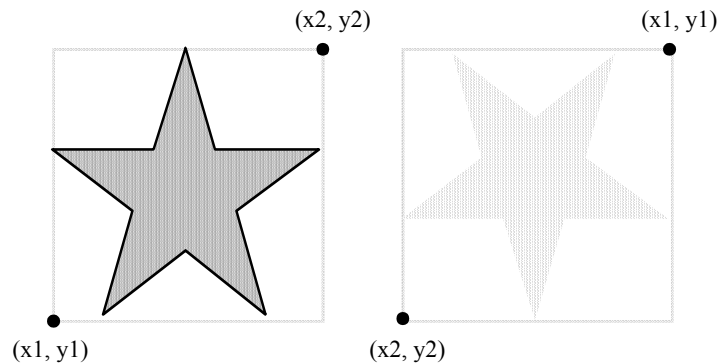
Syntax	Draw.FillPolygon (<i>x, y</i> : array 1 .. * of int, <i>n</i> : int, <i>Color</i> : int)
Description	<p>The Draw.FillPolygon procedure is used to draw a filled polygon with <i>n</i> points. The polygon is described by the points (<i>x</i>(1), <i>y</i>(1)) to (<i>x</i>(2), <i>y</i>(2)) to (<i>x</i>(3), <i>y</i>(3)) and so on to (<i>x</i>(<i>n</i>), <i>y</i>(<i>n</i>)). The polygon will be drawn and filled with <i>Color</i>.</p> <p>To get an polygon outlined in a different color, use Draw.FillPolygon with the <i>Color</i> parameter set to the fill color and then call Draw.Polygon with the <i>Color</i> parameter set to the border color.</p>
Example	<p>This program will create a filled octagon and display it in color 1 and then outline it in color 3.</p> <pre> View.Set ("graphics") var x : array 1..8 of int := init (100, 100, 135, 185, 220, 220, 185, 135) var y : array 1..8 of int := init (100, 150, 185, 185, 150, 100, 65, 65) Draw.FillPolygon (x, y, 8, 1) Draw.Polygon (x, y, 8, 3) </pre>
Details	<p>The PC allows a maximum of 256 points. As well, Draw.FillPolygon can fail (due to lack of memory). If failure occurs, it will try to draw an outline of the polygon. If that also fails, it will not draw anything.</p> <p>The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p>
Status	Exported qualified. This means that you can only call the function by calling Draw.FillPolygon , not by calling FillPolygon .
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.FillStar



Syntax **Draw.FillStar** (*x1, y1, x2, y2, Color : int*)

Description The **Draw.FillStar** procedure is used to draw a filled five pointed star on the screen bounded by a rectangle with bottom left and top right corners of (*x1, y1*) to (*x2, y2*) and filled using the specified *Color*. To get a star outlined in a different color, use **Draw.FillStar** with the *Color* parameter set to the fill color and then call **Draw.Star** with the *Color* parameter set to the border color. If *y1* is greater than *y2*, then the star is drawn upside down.



Example This program will draw two stars beside each other. The first will be outlined in color 1 and filled in color 2. The second star will be upside down and both filled and outlined in color 3.

```
View.Set ( "graphics" )
Draw.FillStar ( 0, 0, 100, 100, 1 )
Draw.Star ( 0, 0, 100, 100, 2 )
Draw.FillStar( 150, 100, 250, 0, 3 )
```

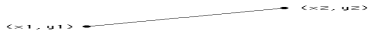
Details The **Draw.FillStar** procedure is useful for drawing the American flag. The screen should be in a "graphics" mode. See the **View.Set** procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.

Status Exported qualified.
This means that you can only call the function by calling **Draw.FillStar**, not by calling **FillStar**.

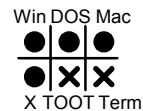
See also **View.Set**, **maxx**, **maxy** and the various procedures in the **Draw** unit.

Draw.Line

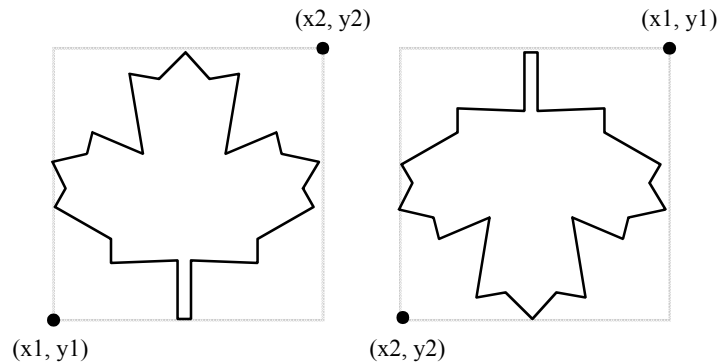


Syntax	Draw.Line (<i>x1, y1, x2, y2, Color : int</i>)
Description	The Draw.Line procedure is used to draw a line on the screen from (<i>x1, y1</i>) to (<i>x2, y2</i>) using the specified <i>Color</i> . 
Example	This program draws a large X, reaching to each corner of the screen using color number 1. The maxx and maxy functions are used to determine the maximum x and y values on the screen. The point (0,0) is the left bottom of the screen, (maxx , maxy) is the right top, etc. <pre> View.Set ("graphics") % First draw a line from the left bottom to right top Draw.Line (0, 0, maxx, maxy, 1) % Now draw a line from the left top to right bottom Draw.Line (0, maxy, maxx, 0, 1) </pre>
Details	The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.
Status	Exported qualified. This means that you can only call the function by calling Draw.Line , not by calling Line .
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.MapleLeaf



Syntax	Draw.MapleLeaf (<i>x1, y1, x2, y2, Color : int</i>)
Description	The Draw.MapleLeaf procedure is used to draw a maple leaf on the screen bounded by a rectangle described by the bottom left and top right corners of (<i>x1, y1</i>) to (<i>x2, y2</i>) using the specified <i>Color</i> . If <i>y1</i> is greater than <i>y2</i> , then the maple leaf is drawn upside down.



Example This program will draw two maple leaves beside each other. The first will be in color 1 and the second maple leaf will be upside down and in color 2.

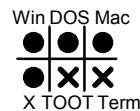
```
View.Set ( "graphics" )
Draw.MapleLeaf ( 0, 0, 100, 100, 1 )
Draw.MapleLeaf ( 150, 100, 250, 0, 2 )
```

Details The **Draw.MapleLeaf** procedure is useful for drawing the Canadian flag. The screen should be in a "graphics" mode. See the **View.Set** procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.

Status Exported qualified.
This means that you can only call the function by calling **Draw.MapleLeaf**, not by calling **MapleLeaf**.

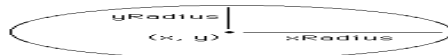
See also **View.Set**, **maxx**, **maxy** and the various procedures in the **Draw** unit.

Draw.Oval



Syntax **Draw.Oval** (*x*, *y*, *xRadius*, *yRadius*, *Color* : **int**)

Description The **Draw.Oval** procedure is used to draw an oval whose center is at (*x*, *y*). The horizontal and vertical distances from the center to the oval are given by *xRadius* and *yRadius*.



Example	<p>This program draws a large oval that just touches each edge of the screen using color number 1. The maxx and maxy functions are used to determine the maximum x and y values on the screen. The center of the oval is at (<i>midx</i>, <i>midy</i>), which is the middle of the screen.</p> <pre> View.Set ("graphics") const <i>midx</i> := maxx div 2 const <i>midy</i> := maxy div 2 Draw.Oval (<i>midx</i>, <i>midy</i>, <i>midx</i>, <i>midy</i>, 1) </pre>
Details	<p>Ideally, a circle is drawn when $xRadius = yRadius$. In fact, the aspect ratio (the ratio of height to width of pixels displayed on the screen) of the IBM PC compatibles is not 1.0, so this does not draw a true circle. In CGA graphics mode this ratio is 5 to 4.</p> <p>The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Draw.Oval, not by calling Oval.</p>
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Draw.Polygon



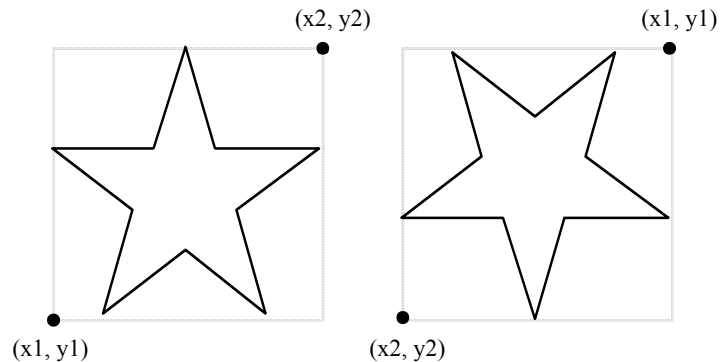
Syntax	Draw.Polygon (<i>x</i> , <i>y</i> : array 1 .. * of int, <i>n</i> : int, <i>Color</i> : int)
Description	<p>The Draw.Polygon procedure is used to draw a polygon with <i>n</i> points. A line is drawn in <i>Color</i> from the point (<i>x</i>(1), <i>y</i>(1)) to (<i>x</i>(2), <i>y</i>(2)) to (<i>x</i>(3), <i>y</i>(3)) and so on. After drawing the line to (<i>x</i>(<i>n</i>), <i>y</i>(<i>n</i>)), a line will be drawn back to (<i>x</i>(1), <i>y</i>(1)), closing the polygon. The Draw.Polygon procedure is equivalent to:</p> <pre> for <i>i</i> : 1 .. <i>n</i> - 1 Draw.Line (<i>x</i> (<i>i</i>), <i>y</i>(<i>i</i>), <i>x</i> (<i>i</i> + 1), <i>y</i> (<i>i</i> + 1), <i>Color</i>) end for Draw.Line (<i>x</i> (<i>n</i>), <i>y</i> (<i>n</i>), <i>x</i> (1), <i>y</i> (1), <i>Color</i>) </pre>
Example	<p>This program will create an octagon and display it in color 1.</p> <pre> View.Set ("graphics") var <i>x</i> : array 1..8 of int := init (100, 100, 135, 185, 220, 220, 185, 135) var <i>y</i> : array 1..8 of int := init (100, 150, 185, 185, </pre>

150, 100, 65, 65)
Draw.Polygon (*x*, *y*, 8, 1)

Details	<p>The IBM PC limits Draw.Polygon to a maximum of 256 points.</p> <p>The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Draw.Polygon, not by calling Polygon.</p>
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Win DOS Mac ●●● ●X X X TOOT Term Draw.Star

Syntax	Draw.Star (<i>x1</i> , <i>y1</i> , <i>x2</i> , <i>y2</i> , <i>Color</i> : int)
Description	<p>The Draw.Star procedure is used to draw a star on the screen bounded by a rectangle described by the bottom left and top right corners of (<i>x1</i>, <i>y1</i>) to (<i>x2</i>, <i>y2</i>) using the specified <i>Color</i>. If <i>y1</i> is greater than <i>y2</i> then the star is drawn upside down.</p>



Example	<p>This program will draw two stars beside each other. The first star will be in color 1 and the second star will be upside down and in color 2.</p> <pre> View.Set ("graphics") Draw.Star (0, 0, 100, 100, 1) Draw.Star (150, 100, 250, 0, 2) </pre>
---------	--

Details	<p>The Draw.Star procedure is useful for drawing the American flag.</p> <p>The screen should be in a "graphics" mode. See the View.Set procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Draw.Star, not by calling Star.</p>
See also	View.Set , maxx , maxy and the various procedures in the Draw unit.

Win DOS Mac ●●○ ●×× X TOOT Tern Draw.Text

Syntax	Draw.Text (<i>txtStr</i> : string , <i>x</i> , <i>y</i> , <i>fontID</i> , <i>Color</i> : int)
Description	<p>Draw.Text is used to actually draw text in a specified font. The <i>txtStr</i> parameter contains the string to be drawn. The <i>x</i> and <i>y</i> paramters are the location of the lower left hand corner of the text to be displayed. The <i>fontID</i> parameter is the number of the font in which the text is to be drawn. The <i>Color</i> parameter is used to specify the color in which the text is to appear.</p> <p>Note that the text that appears is completely unrelated to the text that appears using put. Draw.Text is a graphics command and thus does not use or affect the cursor location.</p> <p>The text drawn by the Draw.Text procedure does not erase the background.</p>
Details	<p>If Draw.Text is passed an invalid font ID, a fatal error occurs. If the Draw.Text call fails for other (non-fatal) reasons then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.</p>
Details	Draw.Text is identical to Font.Draw . It is placed here for consistency with other pixel graphics drawing routines.

Example	<p>The program draws a phrase in red surrounded by a box in bright blue.</p> <pre> var font : int font := Font.New ("serif:12") assert font1 > 0 var width : int := Font.Width ("This is in a serif font", font) var height, ascent, descent, internalLeading : int Font.Sizes (font, height, ascent, descent, internalLeading) Draw.Text ("This is in a serif font", 50, 30, font, red) Draw.Box (50, 30 + descent, 50 + width, 30 + height, brightblue) Font.Free (font) </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Draw.Text, not by calling Text.</p>
See Also	Font module for more information about selecting the font to be displayed.

drawpic graphics procedure Pixel graphics only

Syntax	drawpic (<i>x, y</i> : int , <i>buffer</i> : array 1 .. * of int , <i>picmode</i> : int)
Description	<p>The drawpic procedure is used to copy of a rectangular picture onto the screen. The left bottom of the picture is placed at (<i>x</i>, <i>y</i>). In the common case, the buffer was initialized by calling takepic. The values of <i>picmode</i> are:</p> <ul style="list-style-type: none"> 0: Copy actual picture on screen. 1: Copy picture by XORing it onto the screen. <p>XORing a picture onto the screen twice leaves the screen as it was (this is a convenient way to move images for animation). XORing a picture onto a background effectively superimposes the picture onto the background.</p>
Details	<p>See takepic for an example of the use of drawpic and for further information about buffers for drawing pictures.</p> <p>The screen should be in a "graphics" mode. See the setscreen procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p>
See also	<p>takepic and sizepic. See also setscreen, maxx, maxy and the various draw... procedures.</p> <p>See also predefined unit Draw and Pic.</p>

empty condition function

Syntax	empty (<i>variableReference</i>) : boolean
Description	The empty function is used in a concurrent program. It returns true if the <i>variableReference</i> , which must be a condition variable, has no processes waiting for it. Processes join the queue of a condition variable by executing the wait statement, and are awakened by the signal statement.
See also	condition , wait , signal , fork and monitor . See also predefined unit Concurrency .

enum enumerated type

Syntax	An <i>enumeratedType</i> is: enum (<i>id</i> { , <i>id</i> })
Description	The values of an enumerated type are distinct and increasing. They can be thought of as the values 0, 1, 2 and so on, but arithmetic is not allowed with these values.
Example	<pre>type color : enum (red, green, blue) var c : color := color . red var d : color := succ (c) % d becomes green</pre>
Details	Each value of an enumerated type is the name of the type followed by a dot followed by the element's name, for example, <i>color.red</i> . Enumerated values can be compared for equality and for ordering. The succ and pred functions can be used to find the value following or preceding a given enumerated value. The ord function can be used to find the enumeration position of a value, for example, ord (<i>color.red</i>) is 0. Enumerated types cannot be combined with integers or with other enumerated types.

Details	<p>It is illegal to declare an "anonymous" enum. The only legal declaration for an enum is in a type declaration. For example, the following is now illegal:</p> <pre>var a : array enum (red, green, blue) of int</pre> <p>Given that there is no (easy) way of generating an enum value without it being a named type, this should not impact any but the most bizarre code.</p>
Details	<p>The "put" and "get" statement semantics have been expanded to allow put's and get's of enum values. The values printed and input are the element names themselves, case sensitive. For example, for</p> <pre>type colors : enum (red, green, blue) var c : colors := colors . red put c % outputs "red" (without the quotes)</pre>

enumeratedValue enumerated value

Syntax	<p>An enumeratedValue is:</p> <pre><i>enumeratedTypeId</i> . <i>enumeratedId</i></pre>
Description	<p>The values of an enumerated type are written as the type name (<i>enumeratedTypeId</i>) followed by a dot followed by one of the enumerated values of the type (<i>enumeratedId</i>).</p>
Example	<p>In this example, <i>color.red</i> is an <i>enumeratedValue</i>.</p> <pre>type color : enum (red, green, blue) var c : color := color . red var d : color := succ (c) % d becomes green</pre>
Details	<p>The above description has been simplified by ignoring the possibility that the enum type can be exported from a module. If this is the case, each use of one of the enumerated values outside of module <i>M</i> must be preceded by the module name and a dot, as in <i>M.color.red</i>.</p>
See also	<p>the enum type and <i>explicitConstant</i>.</p>

eof end-of-file function

Syntax **eof** (*streamNumber* : **int**) : **boolean**

Description The **eof** (end of file) function is used to determine if there is any more input. It returns **true** when there are no more characters to be read. The parameter and its parentheses are omitted when referring to the standard input (usually this is the keyboard); otherwise the parameter specifies the number of a stream. The stream number has been determined (in most cases) by an **open** statement.

Example This program reads and outputs all the lines in the file called "info".

```
var line : string
var fileNumber : int
open : fileNumber, "info", get
loop
  exit when eof (fileNumber)
  get : fileNumber, line : *
  put line
end loop
```

Details See also the description of the **get** statement, which gives more examples of the use of **eof**. See also the **open** and **read** statements.

When the input is from the keyboard, the user can signal end-of-file by typing control-Z on a PC (or control-D on UNIX). If a program tests for **eof** on the keyboard, and the user has not typed control-Z (or control-D) and the user has typed no characters beyond those that have been read, the program must wait until the next character is typed. Once this character is typed, the program knows whether it is at the end of the input, and returns the corresponding **true** or **false** value for **eof**.

equivalence of types

Description Two types are *equivalent* to each other if they are essentially the same types (the exact rules are given below). When a variable is passed to a **var** formal parameter, the types of the variable and the formal parameter must be equivalent because they are effectively the same variable. When an expression is assigned to a variable, their types must be equivalent, except for special cases. For example, Turing allows you to assign an integer expression to a **real** variable (see *assignability* for details).

Example

```
var j : int

var b : array 1 .. 25 of string

type personType :
  record
    age : int
    name : string ( 20 )
  end record

procedure p ( var i : int, var a : array 1 .. 25 of string,
              var r : personType )
... body of procedure p, which modifies each of i, a and r ...
end p

var s : personType
p ( j, b, s )      % Procedure call to p
                  % i and j have the equivalent type int
                  % Arrays a and b have equivalent types
                  % Records r and s have equivalent types
```

Details Two types are defined to be *equivalent* if they are:

- (a) the same standard type (**int**, **real**, **boolean** or **string**),
- (b) subranges with equal first and last values,
- (c) arrays with equivalent index types and equivalent component types,
- (d) strings with equal maximum lengths,
- (e) sets with equivalent base types, or
- (f) pointers to the same collection; in addition,
- (g) a declared type identifier is also equivalent to the type it names (and to the type named by that type, if that type is a named type, etc.)
- (h) both **char**,
- (i) both **char**(*n*) with the same length,

- (j) both procedure types, with corresponding equivalent parameter types and corresponding **var** or non-**var** of the parameters,
- (k) both function types, with corresponding equivalent parameter types and corresponding **var** or non-**var** of the parameters and equivalent result types,
- (l) both pointer types to the same class or equivalent type and both are checked or unchecked.

Each separate instance of a record, union or enumerated type (written out using one of the keywords **record**, **union** or **enum**) creates a distinct type, equivalent to no other type. By contrast, separate instances of arrays, strings, subranges and sets are considered equivalent if their parts are equal and equivalent.

Opaque type T , exported from a module, monitor or class M as **opaque**, is a special case of equivalence. Outside of M this type is written $M.T$, and is considered to be distinct from all other types. By contrast, if type U is exported non-**opaque**, the usual rules of equivalence apply. The parameter or result type of an exported procedure or function or an exported constant is considered to have type $M.T$ outside of M if the item is declared using the type identifier T . Outside of M , the **opaque** type can be assigned, but not compared.

It is not required that subprogram types have the same names and parameter names to be equivalent. They also do not require the same factoring of parameters across their types, as in $i, j: \mathbf{int}$ instead of $i: \mathbf{int}, j: \mathbf{int}$.

erealstr real-to-string function

Syntax	erealstr ($r : \mathbf{real}$, $width, fractionWidth, exponentWidth : \mathbf{int}$) : string
Description	<p>The erealstr function is used to convert a real number to a string; for example, erealstr (2.5e1, 10, 3, 2) = "b2.500e+01" where b represents a blank. The string (including exponent) is an approximation to r, padded on the left with blanks as necessary to a length of $width$.</p> <p>The $width$ must be a non-negative int value. If the $width$ parameter is not large enough to represent the value of r, it is implicitly increased as needed.</p>

The *fractionWidth* parameter is the non-negative number of fractional digits to be displayed. The displayed value is rounded to the nearest decimal equivalent with this accuracy. In the case of a tie, the value is rounded to the larger of the two values.

The *exponentWidth* parameter must be non-negative and give the number of exponent digits to be displayed. If *exponentWidth* is not large enough to represent the exponent, more space is used as needed. The string returned by *erealstr* is of the form:

{blank}{[-]digit.{digit}e sign digit {digit}

where *sign* is a plus or minus sign. The leftmost digit is non-zero, unless all the digits are zeros.

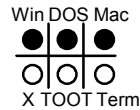
The **erealstr** function approximates the inverse of **streal**, although round-off errors keep these from being exact inverses.

See also **frealstr**, **realstr**, **streal**, **intstr** and **strint** functions.



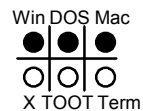
Description	<p>This unit contains the predefined subprograms that deal with errors returned from other predefined subprograms.</p> <p>All routines in the Error unit are exported qualified (and thus must be prefaced with "Error.").</p> <p>The constants representing the possible errors returned by this module can be found in the ErrorNum module.</p>	
Entry Points	Last	Returns the (integer) error code produced by the last call to a predefined subprogram.
	LastMsg	Returns the error string produced by the last call to a predefined subprogram.
	LastStr	Returns the string version of the error constant produced by the last call to a predefined subprogram.
	Msg	Returns the string that corresponds to a specified error code.
	Str	Returns the string version of the error constant that corresponds to a specified error code.
	Trip	This causes execution to halt with returning the specified error.

Error.Last



Syntax	Error.Last : int
Description	<p>Error.Last is a function that returns the error code set by the last called predefined subprogram. If there is no error, then it returns <i>eNoError</i> (which is 0). If there is an error, you can use Error.LastMsg to obtain a textual form of the error or Error.LastStr to obtain a string version of the error constant.</p> <p>The fact that Error.Last is not <i>eNoError</i> does not necessarily mean that the previous predefined function failed or failed completely. Error.Last also returns a number of warning codes. For example, if a user specifies a number larger than maxcolor for the <i>color</i> parameter of the Draw.Line procedure, the line is still drawn, only in color maxcolor. However, Error.Last will return a code that warns the user of the fact.</p>
Example	<p>This program creates the directory called <i>information</i>. If the creation fails, it prints out the error number and an error message.</p> <pre>Dir.Create ("information") if Error.Last = eNoError then put "Directory created" else put "Did not create the directory." put "Error Number: ", Error.Last put "Error Message: ", Error.LastMsg put "Error Constant: ", Error.LastStr end if</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Error.Last, not by calling Last.</p>

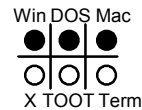
Error.LastMsg



Syntax	Error.LastMsg : string
--------	-------------------------------

Description	<p>Error.LastMsg is a function that returns the error message set by the last called predefined subprogram. If there is no error, then it returns the empty string. If there is an error, you can use Error.Last to obtain the error code.</p> <p>The fact that Error.LastMsg is not "" does not necessarily mean that the previous predefined function failed or failed completely. Error.LastMsg also returns a number of warning messages. For example, if a user specifies a number larger than maxcolor for the <i>color</i> parameter of the Draw.Line procedure, the line is still drawn, only in color maxcolor. However, Error.LastMsg will return a message that indicates that the color was out of range</p>
Example	<p>This program creates the directory called <i>information</i>. If the creation fails, it prints out the error number and an error message.</p> <pre> Dir.Create ("information") if Error.Last = eNoError then put "Directory created" else put "Did not create the directory." put "Error Number: ", Error.Last put "Error Message: ", Error.LastMsg end if </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Error.LastMsg, not by calling LastMsg.</p>

Error.LastStr



Syntax	Error.LastStr : string
Description	<p>Error.LastStr is a function that returns the string version of the error code set by the last called predefined subprogram (i.e. it would return the string "eDrawClrNumTooLarge" for using a color greater than maxcolor in a Draw command). If there is no error then it returns the empty string. If there is an error, you can use Error.Last to obtain the actual error code.</p>

The fact that **Error.LastStr** is not "" does not necessarily mean that the previous predefined function failed or failed completely. **Error.LastStr** also returns a number of error codes for warning messages. For example, if a user specifies a number larger than **maxcolor** for the *color* parameter of the **Draw.Line** procedure, the line is still drawn, only in color **maxcolor**. However, **Error.LastStr** will return a string version of the error code that indicates that the color was out of range.

You can take a look at the error constants defined by looking at the unit **ErrorNum** which contains all defined error codes.

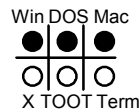
Example This program creates the directory called *information*. If the creation fails, it prints out the error number and an error message.

```
Dir.Create ("information")
if Error.Last = eNoError then
  put "Directory created"
else
  put "Did not create the directory."
  put "Error Number: ", Error.Last
  put "Error Constant: ", Error.LastStr
end if
```

Status Exported qualified.

This means that you can only call the function by calling **Error.LastStr**, not by calling **LastStr**.

Error.Msg



Syntax **Error.Msg** (*errorCode* : int): string

Description **Error.Msg** is a function that returns the error message related to a specified error code. If the error code is *eNoError*, or if there is no such error code, it returns the empty string. If there is such an error, it returns the textual message associated with that error.

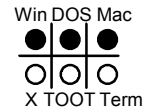
Example This program prints out the error message associated with *eFsysFileNotFound* ("File not found").

```
put Error.Msg (eFsysFileNotFound)
```

Status Exported qualified.

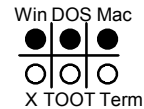
This means that you can only call the function by calling **Error.Msg**, not by calling **Msg**.

Error.Str



Syntax	Error.Str (<i>errorCode</i> : int): string
Description	Error.Str is a function that returns the error message related to a specified error code. If the error code is <i>eNoError</i> or if there is no such error code, it returns the empty string. If there is such an error, it returns the textual message associated with that error.
Example	This program prints out the string "eFsysFileNotFound". put Error.Str (<i>eFsysFileNotFound</i>)
Status	Exported qualified. This means that you can only call the function by calling Error.Str , not by calling Str .

Error.Trip



Syntax	Error.Trip (<i>errorCode</i> : int)
Description	Error.Trip is a procedure that causes execution of the program to abort immediately with an error message that corresponds to the error code passed in. The program will abort even if the error code passed in is normally a warning. Error codes that do not correspond to recognized errors will cause an abort with the error message "Unknown Error #n" where n is the error passed in. You can find a list of constants for the legal error codes in the module ErrorNum . Any call to Error.Trip should use a constant found in the ErrorNum module.
Example	This program aborts execution with the message "File not found". put Error.Trip (<i>eFsysFileNotFound</i>)
Status	Exported qualified.

This means that you can only call the function by calling **Error.Trip**, not by calling **Trip**.

ErrorNum All

Description This unit contains all the constants representing errors used by the Error module.

All constants in the ErrorNum module are exported unqualified. (This means you can use the constants directly without having to use the qualifier "**ErrorNum**".)

Exceptions All

Description This unit contains all the constants corresponding to exception numbers in Turing for use in building exception handlers.

All constants in the Exceptions module are exported unqualified. (This means you can use the constants directly without having to use the qualifier "**Exceptions**".)

exit statement

Syntax An *exitStatement* is one of:

- (a) **exit when** trueFalseExpn
- (b) **exit**

Description	An exit statement is used to stop the execution of a loop or for statement. Form (a) is the most common. Here, the true/false expression is evaluated. If it is true, the loop is terminated and execution jumps down and continues just beyond end loop or end for . If it is false, the loop keeps on repeating. Form (b) always causes the loop to terminate. This form is almost always used inside another conditional statement such as if .
Example	Input names until finding Jones. <pre> var name : string loop get name exit when name = "Jones" end loop </pre>
Details	Exit statements must occur only inside loop or for statements. An exit takes you out of the closest surrounding loop or for . The only other ways to terminate a loop or for is by return (in a procedure or in the main program, in which case the entire procedure or main program is terminated) or by result (in a function, in which case the entire function is terminated and a result value must be supplied). The form " exit when <i>trueFalseExpn</i> " is equivalent to " if <i>trueFalseExpn</i> then exit end if ".

exp exponentiation function

Syntax	exp (<i>r</i> : real) : real
Description	The exp function is used to find e to the power r, where e is the natural base and r is the parameter to exp . For example, exp (0) returns 1 and exp (1) returns the value of e.
Example	This program prints out the exponential values of 1, 2, 3, ... up to 100. <pre> for i : 1 .. 100 put "Exponential of ", i, " is ", exp (i) end for </pre>
See also	ln (natural logarithm function). See also predefined unit Math .

explicitCharConstant character literal

Syntax An *explicitCharConstant* is a sequence of characters surrounded by single quotation marks, for example, 'Renzo'.

Example In the following, the explicit character constants are 'H' and 'Hi'.

```
var c : char := 'H'
      var d : char ( 2 ) := 'Hi'
```

Details An explicit character constant must contain at least one character. If it contains exactly one character, as in 'A', its type is **char**. If it contains two or more characters (*n* characters), as 'Width', its type is **char(n)**. The difference between the **char** and **char(1)** types is rarely of significance, but does make a difference in declarations without an explicit type, for example:

```
var c := 'H'            % Type is char
var d := 'Hi'    % Type is char ( 2 )
      var e := "H"    % Type is string
```

The backslash \ is used in explicit string and char(*n*) constants to specify special values, for example, '\T' is the tab character. Similarly, the carat ^ is used to specify ASCII control characters, for example, '^H' is the ASCII backspace. See *explicitStringConstants* for details.

Explicit character constants cannot cross line boundaries. To represent a constant that is longer than a line, break it into two or more strings on separate lines and use + (catenation) to join the individual strings. See **catenation**.

An explicit character constant may be limited in length by the implementation. We recommend that this limitation be at least 32767.

Explicit character constants, but not strings, are allowed to contain the character internal values 0 (called *eos* for end of string) and 128 (called *uninitchar*, used as the uninitialized string value).

explicitConstant literal

Syntax An *explicitConstant* is one of:

- (a) *explicitStringConstant* % e.g.: "Hello world"
- (b) *explicitIntegerConstant* % e.g.: 25
- (c) *explicitRealConstant* % e.g.: 51.8
- (d) *explicitTrueFalseConstant* % e.g.: **true**
- (e) *explicitCharConstant* % e.g.: 'Hi'

Description	An <i>explicitConstant</i> gives its value directly. For example, the value of the explicit constant 25 is twenty-five.
Example	<p>In the following, the explicit constants are "Hello world", 3.14159 and 2. Note that <i>pi</i> is a <i>named</i> constant rather than an explicit constant.</p> <pre> put "Hello world" var diameter : real const pi := 3.14159 diameter := pi * r ** 2 var x := diameter </pre>
Details	In some programming languages, <i>explicit constants</i> are called <i>literals</i> or <i>literal values</i> , because they literally (explicitly) give their values.
See also	<i>explicitStringConstant</i> , <i>explicitIntegerConstant</i> , <i>explicitRealConstant</i> , <i>explicitTrueFalseConstant</i> and <i>explicitCharConstant</i> . See also <i>enumeratedValue</i> .

explicitIntegerConstant integer literal

Syntax	An <i>explicitIntegerConstant</i> is a sequence of one or more decimal digits (0 to 9) optionally preceded by a plus or minus sign. This is an alternate form that specifies a number base (such as base 2 or base 16).
Example	<p>In the following, the explicit integer constants are 0, 115 and 5.</p> <pre> var count : int := 0 const height := 115 ... count := height - 5 </pre>
Details	In current implementations of Turing, the range of the int (integer) type is from -2147483647 to 2147483647. In other words, the maximum size of integer is $2^{31} - 1$. This is the range that fits into four bytes, with one pattern left over (the largest negative 4-byte number) to represent the uninitialized value. See maxint .

Values can be written in base 2 or 16 or any other base in the range 2 to 36 (36 because there are 10 digits and 26 letters). This form begins with the base, such as 16, then #, and then the value written in that base, for example, 16#A has the value 10. The letters a, b, c ... represent the digit values 10, 11, 12 ... Capital letters A, B, C ... can be used instead of lower case. Here are some examples.

2#1	= 1	(Base 2)
2#11	= 3	(Base 2)
16#a	= 10	(Base 16)
16#FF	= 255	(Base 16)
16#FFFF	= 32767	(Base 16)
8#10	= 8	(Base 8)

Here is an example of using these:

```
const maxnat1 := 16#FF      % Largest 1-byte natural number
const maxint2 := 16#7FFF    % Largest 2-byte integer
```

You should be careful to avoid confusion about patterns such as 16#FFFF. It is tempting to think that this is the value -1, because the bit pattern (2-byte two's complement internal representation) for -1 is the same as the bit pattern for 16#FFFF = 32767. However, the value (as opposed to the internal representation) of -1 and 32767 are different.

See also **int**, **maxint** (the largest integer value), **nat** (positive values only) and **int*n*** (*n*-byte integers). See also **intstr** and **natstr** which convert integer and natural number values to corresponding character strings in any base, for example **intstr** (4, 0, 2) = "100".

explicitRealConstant real literal

Syntax An *explicitRealConstant* consists of an optional plus or minus sign, a *significant digits part*, and an *exponent part*.

Example In the following, the explicit real constants are 0.0 and 2.93e3.

```
var temperature : real := 0.0
const speed := 2.93e3      % Value is 2,930.0
```

Details The significant digits part (or *fractional part*) of an explicit real constant consists of a sequence of one or more digits (0 to 9) optionally containing a decimal point (a period). The decimal point is allowed to follow the last digit as in 16. or to precede the first digit, as in .25.

The exponent part consists of the letter *e* or *E* followed optionally by a plus or minus sign followed by one or more digits. For example, in `-9.837e-3` the exponent part is `e-3`. The value of `-9.837e-3` is `-9.837` times `0.001`.

If the significant figures part contains a decimal point, then the exponent part is not required.

explicitStringConstant string literal

Syntax	An <i>explicitStringConstant</i> is a sequence of characters surrounded by quotation marks.
Example	In the following, the explicit string constants are <code>"Hello world"</code> , <code>""</code> and <code>"273 O'Reilly Ave."</code> . <pre>var name : string := "Hello world" name := "" % Null string, containing zero characters var address : string := "273 O'Reilly Ave."</pre>

Details	Within an explicit string constant (and within an explicit character constant), the back slash <code>\</code> is used to represent certain other characters as follows:
---------	---

<code>\ "</code>	quotation mark character
<code>\n</code> or <code>\N</code>	end of line character
<code>\t</code> or <code>\Tab</code>	tab character
<code>\f</code> or <code>\F</code>	form feed character
<code>\r</code> or <code>\R</code>	return character
<code>\b</code> or <code>\B</code>	backspace character
<code>\e</code> or <code>\E</code>	escape character
<code>\d</code> or <code>\D</code>	delete character
<code>\\</code>	backslash character

For example, `put "One\nTwo"` will output *One* on one line and *Two* on the next. In an explicit character constant (which is surrounded by single quotes, as in `'John'`), the backslash is not required before a double quote `"`, but it is required before a single quote `'`, as in these two constants:

```
'John said "Hello" to you'
'Don\'t cry'.
```

You can use the caret `^` to specify ASCII control characters, for example:

'^H' ASCII backspace character

The caret specifies that the top three bits of the character are set to zero. For any character *c*, the following is true:

'^c' = chr (ord ('c') & 2#11111)

However if *c* is the question mark, as in '^?', the bits are not turned off.

Explicit string constants cannot cross line boundaries. To represent a string that is longer than a line, break it into two or more strings on separate lines and use catenation (+) to join the individual strings.

An explicit string constant can contain at most 255 characters (this is in implementation constraint).

String values are not allowed to contain characters with the code values of 0 or 128; these character values are called *eos* (end of string) and *uninitchar* (uninitialized character). These are reserved by the implementation to mark the end of a string value and to see if a string variable has been initialized.

explicitTrueFalseConstant boolean literal

Syntax An explicitTrueFalseConstant is one of:

- (a) **true**
- (b) **false**

Example The following determines if string *s* contains a period. After the **for** statement, *found* will be **true** if there is a period in *s*.

```
var found : boolean := false
for i : 1 .. length ( s )
  if s = "." then
    found := true
  end if
end for
```

Details **true/false** values are called *boolean* values. A **boolean** variable, such as *found* in the above example, can have a value of either **true** or **false**.

See also **boolean** type.

expn expression

Syntax	<p>An <i>expn</i> is one of:</p> <ul style="list-style-type: none"> (a) <i>explicitConstant</i> % e.g.: 25 (b) <i>variableReference</i> % e.g.: <i>width</i> (c) <i>constantReference</i> % e.g.: <i>pi</i> (d) <i>expn infixOperator expn</i> % e.g.: 3 + <i>width</i> (e) <i>prefixOperator expn</i> % e.g.: - <i>width</i> (f) (<i>expn</i>) % e.g.: (<i>width</i> - 7) (g) <i>substring</i> % e.g.: <i>s</i> (3 .. 5) (h) <i>functionCall</i> % e.g.: <i>sqrt</i> (25) (i) <i>setConstructor</i> % e.g.: <i>modes</i> (4, 3) (j) <i>enumeratedValue</i> % e.g.: <i>color</i> . <i>red</i>
Description	<p>An expression (<i>expn</i>) returns a value; in the general case, this may involve a calculation, such as addition, as in the expression:</p> <p style="text-align: center;">3 + <i>width</i></p>
Example	<pre> put "Hello world" % "Hello world" is an expn var diameter : real const pi := 3.14159 % 3.14159 is an expn diameter := pi * r ** 2 % pi * r ** 2 is an expn var x := diameter % diameter is an expn </pre>
Details	<p>In the simplest case, an expression (<i>expn</i>) is simply an explicit constant such as 25 or "Hello world". A variable by itself is considered to be an expression when its value is used. This is the case above, where the value of <i>diameter</i> is used to initialize <i>x</i>. More generally, an expression contains an operator such as + and carries out an actual calculation. An expression may also be a substring, function call, set constructor or enumerated value. For details, see the descriptions of these items.</p> <p>The Turing infix operators are: +, -, *, /, div, mod, **, <, >, =, <=, >=, not=, not, and, or, =>, in, not in, shr (shift right), shl (shift left), and xor (exclusive or). For details, see <i>infixOperator</i>. The Turing prefix operators are +, - and not, ^ (pointer following) and # (see cheat). For details see <i>prefix operator</i>.</p>
See also	<p><i>precedence</i> of operators, as well as the int, real, string and boolean types.</p>

export list

Syntax	<p>An <i>exportList</i> is:</p> <p>export [<i>howExport</i>] <i>id</i> {, [<i>howExport</i>] <i>id</i> }</p>
Description	<p>An export list is used to specify those items declared in a module, monitor or class that can be used outside of it. Items that are declared inside a module, monitor or class but not exported cannot be accessed outside of it.</p>
Example	<p>In this example, the procedures names <i>pop</i> and <i>push</i> are exported from the <i>stack</i> module. These two procedures are called from outside the module on the last and third from last lines of the example. Notice that the word <i>stack</i> and a dot must precede the use of these names. Since <i>top</i> and <i>contents</i> were not exported, they can be accessed only from inside the module.</p> <pre>module stack export push, pop var top : int := 0 var contents : array 1..100 of string procedure push ... end push procedure pop ... end pop end stack stack . push ("Harvey") var name : string stack . pop (name) % This sets name to Harvey</pre>
Details	<p>Procedures, functions, variables, constants and types can be exported. Modules, monitors or classes cannot be exported. Parentheses are allowed around the items in an export list, as in:</p> <p>export (<i>push</i>, <i>pop</i>)</p> <p>The following syntax specifies that each exported identifier can optionally be preceded by the keywords var, unqualified, pervasive and opaque. Of these, only opaque is available in Turing proper.</p> <p>The form of <i>howExport</i> is:</p> <p>{ <i>exportMethod</i> }</p> <p>The form of <i>exportMethod</i> is one of:</p> <ul style="list-style-type: none">(a) var(b) unqualified(c) pervasive(d) opaque

The keyword **var** means that the exported variable can be changed outside of the exporting module, monitor or class. This keyword applies only to exported variables. For example, if string variable *name* is exported **var** from module *M*, *name* can be changed from outside of *M* by *M.name* := "Surprise!".

The keyword **unqualified** means that references to the exported item do not need to be prefixed by the name of the exporting item. For example, if module *M* exports procedure *p* unqualified, a call to *p* outside of *M* can be simply *p* instead of the usual *M.p*. A class cannot export variables or dynamic constants unqualified (because each object of the class has its own copies of these). The only things a class can export unqualified are types and compile time constants. The keyword **unqualified** can be abbreviated to **~.** which is pronounced as "not dot".

The keyword **pervasive**, which is only meaningful if **unqualified** is also present, specifies that the exported item is to be visible in subsequent scopes, in other words that it is not necessary to import it into internal modules, monitors and classes.

The keyword **opaque**, which can only precede type names, specifies that outside the module, monitor or class, the type is considered to be distinct from all other types. This means, for example, that if the type is an array, it cannot be subscripted outside of the module. See **module** declaration for an example that uses opaque types. In most cases, classes are preferable to opaque types.

Exported subprograms are considered to be **deferred**, meaning that expansions are allowed to override these subprograms. See also **deferred** subprograms. These can be overridden using the keyword **body** before the resolving subprogram body.

A class cannot export items from its parent or its parent's ancestors. All exported items must be declared in the current class.

Details

You can export **all** from a module, monitor or a class. This means that every symbol that is legal to export is exported. You may also qualify the **all**, as in **export opaque unqualified pervasive all** where the qualifiers are added to each export item (if it makes sense).

If **all** is specified as the export item, no other item may be specified. Also, and **all** export affects only the module, monitor or class that it is given in. Any inheriting or implementing module, monitor or class does not export **all** unless they also specify it.

See also

unit, **module**, **monitor** and **class**. See also **import** list, **inherit** clause, **implement** clause, **implement by** clause and **deferred** subprogram.

external declaration

Dangerous

Syntax	<p>An <i>externalDeclaration</i> is one of:</p> <p>(a) external [<i>overrideName</i>] <i>subprogramHeader</i></p> <p>(b) external [<i>addressSpec</i>] var <i>id</i> [: <i>typeSpec</i>] [:=<i>expn</i>]</p>
Description	<p>An external declaration is used to access variables or subprograms that are written in other languages or which require special linkage. This feature is implementation-dependent and dangerous and may cause arbitrary data or program corruption. From an interpretive environment such as Turing, this provides linkage to items that are part of the Turing system. For compiled versions of Turing, the linkage would be by means of a standard, operating system-specific linkage editor.</p>
Details	<p>In form (a) the optional <i>overrideName</i> must be an explicit string constant, such as <i>"printf"</i>. If it is omitted, the external name is the name in the <i>subprogramHeader</i>. See <i>subprogramHeader</i>.</p> <p>The current implementation does not support form (b). This form is documented here in case a future version supports it. The <i>addressSpec</i> is a compile time expression (its value must fit in the range of the addressint type) or is a compile time string value. If the <i>addressSpec</i> is omitted, the identifier is the name of an external variable. This name represents an implementation-dependent method of locating a variable. At least one of <i>typeSpec</i> or <i>expn</i> must be present.</p> <p>Declaring variables at absolute addresses is useful for device management in computer architectures with memory mapped device registers. External variables declared to be int or nat will by default be checked for initialization. To avoid this check, declare them to be int4 or nat4.</p>
Example	<p>Place variable <i>ttyData</i> at hexadecimal location 9001 and assign it the character A.</p> <pre>external 16#9001 var <i>ttyData</i> : char <i>ttyData</i> := 'A' % Character A is assigned to hex location 9001</pre>
Example	<p>Access an external integer variable named ERRFLAG.</p> <pre>external var ERRFLAG : int if ERRFLAG = 0 then ...</pre>
Example	<p>Access an integer variable which is called <i>y</i> in this program but is called <i>x</i> externally.</p> <pre>external "x" var <i>y</i> : int</pre>

Example Declare *drawcircle* to be a procedure that is externally known as *circle*.
external "circle" **procedure** *drawcircle* (*x*, *y*, *r*, *Color* : **int**)

false boolean value (not true)

Syntax **false**

Description A **boolean** (**true/false**) variable can be either **true** or **false** (see **boolean** type).

Example

```
var found : boolean := false
var word : int
for i : 1 .. 10
    get word
    found := found or word = "gold"
end for
if found = true then
    put "Found 'gold' in the ten words"
end if
```

Details The line **if** *found*=**true** **then** can be simplified to **if** *found* **then** with no change to the meaning of the program.

fetcharg fetch argument function

Syntax **fetcharg** (*i* : **int**) : **string**

Description The **fetcharg** function is used to access the *i*-th argument that has been passed to a program from the command line. For example, if the program is run from the Turing environment using

:r file1 file2

then **fetcharg**(2) will return "file2". If a program called *prog.x* is run under UNIX using this command:

prog.x file1 file2

the value of **fetcharg**(2) will similarly be "file2".

The **nargs** function, which gives the number of arguments passed to the program, is usually used together with the **fetcharg** function. Parameter *i* passed to **fetcharg** must be in the range 0 .. **nargs**.

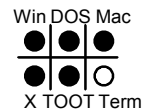
The 0-th argument is the name of the running program.

Example This program lists its own name and its arguments.

```
put "The name of this program is : ", fetcharg ( 0 )
for i : 1 .. nargs
  put "Argument ", i, " is ", fetcharg ( i )
end for
```

See also **nargs**

File



Description This unit contains the predefined subprograms that deal with file manipulation on a whole-file basis (as opposed to manipulating the data in the file using **open** and **close**, etc.). These routines allow you to rename, copy and delete files, as well as get information about a file and get the free space on disk available for a file.

All routines in the File module are exported qualified (and thus must be prefaced with "**File.**").

Entry Points	Exists	Returns whether a file exists.
	Status	Gets information about a file such as size, modification date, etc.
	Copy	Copies a file to another location.
	Rename	Renames a file or directory.
	Delete	Deletes a file.
	DiskFree	Gets the free space on the disk upon which a file or directory resides.

Details On the PC, a path name of a file or a directory can use either the forward slash or backward slash to separate directory names. The drive must be followed by a colon. Thus the following are legal path names:

```
x:\students\west\example.t
c:/turing/test.t
```


/west/binary.t (uses the default drive).

On the Macintosh, a path name of a file or directory can use the standard Macintosh format of Volume Name:Directory Name:Directory Name:File Name or the Unix format of /Volume Name/Directory Name/Directory Name/File Name. Note that the names can have spaces in them.

HSA:Applications:Turing Files:example.t
/HSA/Applications/Turing Files/example.t

On UNIX systems, the path name must correspond to the UNIX standard of using a forward slash between parts of the path.

/export/home/west/turing/example.t

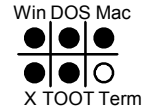
In general, you can achieve the greatest portability by using the UNIX standard for use in path names, as all Turing systems support it.



Syntax	File.Copy (<i>srcPathName</i> , <i>destPathName</i> : string)
Description	File.Copy copies a file named by the <i>srcPathName</i> parameter to the file named by the <i>destPathName</i> parameter. The copy can be between different disks or file systems.
Details	The source file name must be an actual file. This procedure will not copy directories. If the File.Copy call fails, then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.
Example	This program copies the file "/usr/west/example" to "/usr/holt/testcase". <pre>File.Copy ("/usr/west/example", "/usr/holt/testcase") if Error.Last = <i>eNoError</i> then put "File copied" else put "Did not copy the file." put "Error: ", Error.LastMsg end if</pre>
Status	Exported qualified.

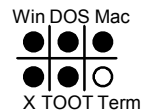
This means that you can only call the function by calling **File.Copy**, not by calling **Copy**.

File.Delete



Syntax	File.Delete (<i>filePathName</i> : string)
Description	File.Delete is used to delete the file specified by the parameter <i>filePathName</i> . This is the equivalent of doing a del in DOS or rm in UNIX.
Details	If the File.Delete call fails, then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.
Example	<p>This program deletes the file called <i>information</i>.</p> <pre>File.Delete ("information") if Error.Last = <i>eNoError</i> then put "File delete" else put "Did not delete the file." put "Error: ", Error.LastMsg end if</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling File.Delete, not by calling File.</p>

File.DiskFree



Syntax	File.DiskFree (<i>pathName</i> : string) : int
Description	File.DiskFree gets the number of bytes for the disk upon which <i>pathName</i> resides. The <i>pathName</i> parameter can specify either a file or a directory. If it is the empty string, then File.DiskFree returns the number of bytes of free disk space on the disk upon which the execution directory resides.

Details If the **File.DiskFree** call fails, then it returns -1. Also **Error.Last** will return a non-zero value indicating the reason for the failure. **Error.LastMsg** will return a string which contains the textual version of the error.

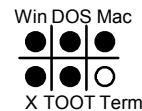
Example This program prints out the amount of space on the A: drive on a PC and in the execution directory.

```
var bytesFree : int
bytesFree := File.DiskFree ("A:\\")
if bytesFree = -1 then
    put "Can't get free space on drive A:."
    put "Error: ", Error.LastMsg
else
    put "There are ", bytesFree , " bytes free on drive A:"
end if

bytesFree := File.DiskFree ("")
if bytesFree = -1 then
    put "Can't get free space on default directory."
    put "Error: ", Error.LastMsg
else
    put "There are ", bytesFree , " bytes free on the default dir"
end if
```

Status Exported qualified.
This means that you can only call the function by calling **File.DiskFree**, not by calling **DiskFree**.

File.Exists



Syntax **File.Exists** (*pathName* : **string**) : **boolean**

Description **File.Exists** returns **true** if a file by the name of *pathName* exists. It will return **true** if *pathName* is a file or a directory.

Details If the **File.Exists** returns **false**, you can examine **Error.Last** for more information (i.e. whether the path failed or the file was simply not found).

Example This program loops until the user types in a path name that either doesn't already exist or is allowed to be overwritten.

```
var pathName : string
var choice : string
loop
    put "Enter file name to write results to" ..
    get pathName
```

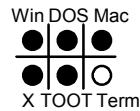
```

if File.Exists (pathName) then
    put "Overwrite ", pathName, "?" ..
    get choice
    exit when choice = "y"
else
    exit
end if
end loop

```

Status Exported qualified.
This means that you can only call the function by calling **File.Exists**, not by calling **Exists**.

File.Rename



Syntax **File.Rename** (*srcPathName*, *destName* : **string**)

Description **File.Copy** renames a file or directory named by the *srcPathName* parameter to the *destName* parameter. The *destName* parameter must be a name only. In other words **File.Rename** can't move a file between different directories.

Details If the **File.Rename** call fails, then **Error.Last** will return a non-zero value indicating the reason for the failure. **Error.LastMsg** will return a string which contains the textual version of the error.

Example This program renames the file `"/usr/west/example"` to `"testcase"`

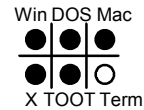
```

File.Rename ("/usr/west/example", "testcase")
if Error.Last = eNoError then
    put "File renamed"
else
    put "Did not rename the file."
    put "Error: ", Error.LastMsg
end if

```

Status Exported qualified.
This means that you can only call the function by calling **File.Rename**, not by calling **Rename**.

File.Status



Syntax	File.Status (<i>pathName</i> : string , var <i>size</i> , <i>attribute</i> , <i>fileTime</i> : int)
Description	<p>File.Status is used to get assorted information about a file or directory. When the function is called with a specified <i>pathName</i>, it returns the information about the file in the other parameters.</p> <p>The <i>size</i> parameter is the size of the file in bytes.</p> <p>The <i>attribute</i> parameter has its individual bits set as exactly as the <i>attribute</i> parameter in Dir.GetLong subprogram does. See Dir.GetLong for the list of attribute constants.</p> <p>The <i>fileTime</i> is the time of last modification of the file. It is returned as the number of seconds since 00:00:00 GMT 1/1/1970. To convert this to a string, use Time.SecDate.</p>
Details	<p>If the File.Status call fails, <i>size</i>, <i>attribute</i> and <i>fileTime</i> are all set to -1.</p> <p>Error.Last will return a non-zero value indicating the reason for the failure.</p> <p>Error.LastMsg will return a string which contains the textual version of the error.</p>
Example	<p>This program prints an approximation of the UNIX "ls -l" command for the file "~/mail".</p>

```

const pathName : string := "~/mail"
var size, attribute, fileTime : int
File.Status (pathName, size, attribute, fileTime )

if Error.Last = eNoError then
  put pathName, " ", Time.SecDate (fileTime)
  if (attribute and ootAttrDir) not= 0 then
    put "d" ..
  else
    put "-" ..
  end if
  if (attribute and ootAttrOwnerRead) not= 0 then
    put "r" ..
  else
    put "-" ..
  end if
  if (attribute and ootAttrOwnerWrite) not= 0 then
    put "w" ..
  else
    put "-" ..
  end if
  if (attribute and ootAttrOwnerExecute) not= 0 then
    put "x" ..
  end if
end if

```

```

else
  put "-" ..
end if
if (attribute and ootAttrGroupRead) not= 0 then
  put "r" ..
else
  put "-" ..
end if
if (attribute and ootAttrGroupWrite) not= 0 then
  put "w" ..
else
  put "-" ..
end if
if (attribute and ootAttrGroupExecute) not= 0 then
  put "x" ..
else
  put "-" ..
end if
if (attribute and ootAttrGroupRead) not= 0 then
  put "r" ..
else
  put "-" ..
end if
if (attribute and ootAttrGroupWrite) not= 0 then
  put "w" ..
else
  put "-" ..
end if
if (attribute and ootAttrGroupExecute) not= 0 then
  put "x" ..
else
  put "-" ..
end if
put size : 8, " ", Time.SecDate (fileTime), " ", pathName
else
  put "Unable to get file information"
  put "Error: ", Error.LastMsg
  end if

```

Status Exported qualified.

 This means that you can only call the function by calling **File.Status**, not by calling **Status**.

flexible array initialization

Syntax **flexible array** *indexType* { , *indexType* } **of** *typeSpec*

Description	<p>The flexible keyword allows an array to be resized using new at a later point in time. The indices may have compile-time or run-time upper bounds (the lower bound must be compile-time). The upper bounds can be changed by using:</p> <pre>new name , newUpper1 {,newUpper2}</pre> <p>The existing array entries will retain their values, except that any index made smaller will have the corresponding array entries lost. Any index made larger will have the new array entries uninitialized (if applicable). Additionally, the upper bound (both in the declaration and the new statement) can be made one less than the lower bound. This effectively makes an array that contains 0 elements. It can later be increased in size with another new.</p> <p>In the current implementation (1999), with a multi-dimensional array with a non-zero number of total elements, it is a run-time error to change any but the first dimension (unless one of the new upper bounds is one less than the corresponding lower bound, giving 0 elements in the array) as the algorithm to rearrange the element memory locations has not yet been implemented.</p> <p>Currently, only variables can be declared in this form. There is no flexible array parameter type, although a flexible array can be passed to an array parameter with "*" as the upper bound.</p>
Example	See array for an example of flexible .
See also	array and new .

floor real-to-integer function

Syntax	floor (<i>r</i> : real) : int
Description	Returns the largest integer that is less than or equal to <i>r</i> .
Details	The floor function is used to convert a real number to an integer. The result is the largest integer that is less than or equal to <i>r</i> . In other words, the floor function rounds down to the nearest integer. For example, floor (3) is 3, floor (2.75) is 2 and floor (-8.43) is -9.
See also	ceil and round functions.

Font

Description	<p>This unit contains the predefined subprograms that deal with fonts. Using these routines, you can display text in a selected font name, size and style on the screen. Note that output in a particular font is treated as graphics output.</p> <p>All routines in the Font module are exported qualified (and thus must be prefaced with "Font.").</p>	
Details	<p>There is a default font. You can draw in and obtain information about the default font by passing <i>fontDefaultID</i> to Font.Draw, Font.Width and Font.Sizes. The default font is the same font as is used by put in the output window.</p>	
Entry Points	New	Selects a particular font name, size and style for a new font.
	Free	Frees up the font created by using New .
	Draw	Draws text in a given font.
	Width	Gets the width in pixels of a particular piece of text in a specified font.
	Sizes	Gets the height and various leadings of a specified font.
	Name	Returns the name of the specified font.
	StartName	Prepares to list all available fonts,
	GetName	Gets the next font name.
	GetStyle	Gets all the available styles for a specified font.
	StartSize	Prepares to list all available sizes for a specified font and style.
	GetSize	Gets the next font size.

Font.Draw



Syntax	Font.Draw (<i>txtStr</i> : string , <i>x</i> , <i>y</i> , <i>fontID</i> , <i>Color</i> : int)
Description	<p>Font.Draw is used to actually draw text in a specified font. The <i>txtStr</i> parameter contains the string to be drawn. The <i>x</i> and <i>y</i> parameters are the location of the lower left hand corner of the text to be displayed. The <i>fontID</i> parameter is the number of the font in which the text is to be drawn. The <i>Color</i> parameter is used to specify the color in which the text is to appear.</p> <p>Note that the text that appears is completely unrelated to the text that appears using put. Font.Draw is a graphics command and thus does not use or affect the cursor location.</p> <p>The text drawn by the Font.Draw procedure does not erase the background.</p>
Details	<p>If Font.Draw is passed an invalid font ID, a fatal error occurs. If the Font.Draw call fails for other (non-fatal) reasons, then Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.</p>
Example	<p>The program prints out several phrases in a variety of fonts.</p> <pre> var font1, font2, font3, font4 : int font1 := Font.New ("serif:12") assert font1 > 0 font2 := Font.New ("sans serif:18:bold") assert font2 > 0 font3 := Font.New ("mono:9") assert font3 > 0 font4 := Font.New ("Palatino:24:bold,italic") assert font4 > 0 Font.Draw ("This is in a serif font", 50, 30, font1, red) Font.Draw ("This is in a sans serif font", 50, 80, font2, brightblue) Font.Draw ("This is in a mono font", 50, 130, font3, colorfg) Font.Draw ("This is in Palatino (if available)", 50, 180, font4, green) Font.Free (font1) Font.Free (font2) Font.Free (font3) Font.Free (font4) </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Font.Draw, not by calling Draw.</p>

Font.Free



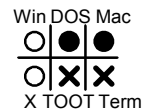
Syntax	Font.Free (<i>fontID</i> : int)
Description	Font.Free is used to release a font that is no longer needed. There is a limit to the number of fonts that may be defined at any one time. By having a Font.Free for every Font.New , the number of simultaneously defined fonts is kept to a minimum.
Details	If Font.Free is passed an invalid font ID, a fatal error occurs. If the Font.Free call fails for other (non-fatal) reasons, Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.
Example	<p>The program prints out several phrases in a variety of fonts.</p> <pre>var font1, font2, font3, font4 : int font1 := Font.New ("serif:12") assert font1 > 0 font2 := Font.New ("sans serif:18:bold") assert font2 > 0 font3 := Font.New ("mono:9") assert font3 > 0 font4 := Font.New ("Palatino:24:Bold,Italic") assert font4 > 0 Font.Draw ("This is in a serif font", 50, 30, font1, red) Font.Draw ("This is in a sans serif font", 50, 80, font2, brightblue) Font.Draw ("This is in a mono font", 50, 130, font3, colorfg) Font.Draw ("This is in Palatino (if available)", 50, 180, font4, green) Font.Free (font1) Font.Free (font2) Font.Free (font3) Font.Free (font4)</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Font.Free, not by calling Free.</p>

Font.GetName



Syntax	Font.GetName : string
Description	<p>Font.GetName is used to get the next font available on the system. By using Font.StartName and then calling Font.GetName repeatedly, you can get the names of all the fonts available to the program.</p> <p>Font.StartName must be called before any calls to Font.GetName. After that, Font.GetName returns the list of the font names, one per call. When there are no more sizes, Font.GetName returns the empty string.</p> <p>Once the name of a font is known, it's possible to list the available styles (using Font.GetStyle) and the available sizes (using Font.StartSize and Font.GetSize) for that font.</p>
Example	<p>The program lists all the fonts available on the system.</p> <pre>var fontName : string Font.StartName loop fontName := Font.GetName exit when fontName = "" put fontName end loop</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Font.GetName, not by calling GetName.</p>

Font.GetSize



Syntax	Font.GetSize : int
Description	<p>Font.GetSize is used to get the next size in the list of available font sizes for a particular font name and style.</p> <p>Font.StartSize must be called before any calls to Font.GetSize. After that, Font.GetSize returns the list of sizes, one per call. When there are no more sizes, Font.GetSize returns 0.</p>

Some fonts are “scalable”. This means that the computer can scale the fonts to fit any given size. (Under Microsoft Windows and the Apple Macintosh, TrueType and PostScript fonts are scalable with the appropriate utilities.) In this case, **Font.GetSize** returns -1.

Example	See Font.StartSize for a program that lists all the fonts, styles and sizes available on the system.
Status	Exported qualified. This means that you can only call the function by calling Font.GetSize , not by calling GetSize .

Font.GetStyle

Win	DOS	Mac
○	●	●
○	×	×
X TOOT Term		

Syntax **Font.GetStyle** (*fontName* : **string**,
 bold, italic, underline : **boolean**): **string**

Description **Font.GetStyle** is used to get the styles available on the system for a specified font. *bold*, *italic* and *underline* are set to **true** if bold, italic or underline versions of the font are available. Once the styles available for a font are known, it's possible to get the sizes available for each style by using **Font.StartSize** and **Font.GetSize**.

Example The program lists all the fonts and their styles available on the system.

```

var fontName : string
var bold, italic, underline : boolean
Font.StartName
loop
  fontName := Font.GetName
  exit when fontName = ""
  Font.GetStyle (fontName, bold, italic, underline)
  put fontName : 30 ..
  if bold then
    put "bold " ..
  end if
  if italic then
    put "italic " ..
  end if
  if underline then
    put "underline " ..
  end if
end loop

```

Status Exported qualified.

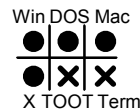
This means that you can only call the function by calling **Font.GetStyle**, not by calling **GetStyle**.

Font.Name



Syntax	Font.Name (<i>fontID</i> : int) <i>fontName</i> : string
Description	Font.Name is used to get the name of a font that is being used. The string that is returned can be used to determine which font is actually being used for the default fonts “serif”, “sans serif” and “mono”.
Example	<p>The program prints out the fonts used for “serif”, “sans serif” and “mono”.</p> <pre>var serifFont, sansSerifFont, monoFont : int serifFont := Font.New ("serif:12") assert serifFont > 0 sansSerifFont := Font.New ("sans serif:12") assert sansSerifFont > 0 monoFont := Font.New ("mono:12") assert monoFont > 0 put "serif = ", Font.Name (serifFont) put "sans serif = ", Font.Name (sansSerifFont) put "mono = ", Font.Name (monoFont) Font.Free (serifFont) Font.Free (sansSerifFont) Font.Free (monoFont)</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Font.Name, not by calling Name.</p>

Font.New



Syntax	Font.New (<i>fontSelectStr</i> : string) : int
--------	---

Description	<p>Font.New is used to obtain a font for drawing. The <i>fontSelectStr</i> parameter specifies the name, size and style of the font. Font.New returns a font identifier which is then used by the Font.Draw procedure to draw text in the selected font.</p> <p>The format for the <i>fontSelectStr</i> parameter is "<i>Family:Size:Style</i>". Each element is separated by a colon. The "<i>:Style</i>" is optional. If left out, the text appears in the standard face for the font.</p> <p><i>Family</i> is the name of the font, such as "Times", "Helvetica", etc. The name must match an existing font on the system. Because one does not necessarily know which fonts will be available and names for the same font change between different systems (i.e Times, Times-Roman, etc.), OOT defines three family names that will be mapped as closely as possible to fonts that exist on the system.</p> <ul style="list-style-type: none"> • "serif" is used for a serified body font. This will usually be mapped to Times-Roman. • "sans serif" is used for a non-serified display font. This will usually be mapped to Helvetica. • "mono" is used for a mono spaced font. This will usually be mapped to Courier. <p><i>Size</i> is the point size in which the text should appear. If the number is larger or smaller than can be created on a given system, the system will return the font of the largest or smallest size available and set Error.Last.</p> <p>Under WinOOT, the <i>size</i> parameter may also have the form <i>height</i> x <i>width</i> where <i>height</i> and <i>width</i> are the pixel height and width desired. What is returned is the font scaled in order to fit into the <i>width</i> and <i>height</i> requested. The font name must be a scaleable font for this to succeed.</p> <p style="padding-left: 40px;">example <i>fontID</i> := Font.New ("Ariel:18x12:Italic")</p> <p><i>Style</i> is the font style in which the text should appear. It can be one of "bold", "italic" or "underline". You can also have "bold,italic" and any other combination.</p>
Details	<p>The DOS OOT font module uses Microsoft Windows font files (those labelled with the ".FON" file name suffix. If you don't have MS Windows installed and thus have no font files, you will be limited to two fonts detailed below.</p>
In order to find	<p>the font file, DOS OOT must be provided with the name of the directory that the ".FON" files are located in. By default, this is the "C:\WINDOWS\SYSTEM" directory. If the fonts are located there, nothing further need be done. However, if they are located in a different directory, then the directory must be specified by using the "-fontdir=" start up option. The directory is specified either as a command line argument or in the "turing.cfg" file (see Chapter Four) as "-fontdir=<dirname>". For example, if one had all the fonts placed in the directory D:\FONTS, you would start DOS OOT with:</p> <p style="text-align: center;">DOSOOT -fontdir=D:\FONTS</p>

There are two fonts that are always available, regardless of the presence or absence of MS Window fonts. These are the default font (font number 0) and a font called “vector”. In DOS OOT, the default font is 8 pixels in height and width. The “vector” font is a completely scalable font. To use it, you must specify both the height and the width of the font when calling **Font.New**. The Font is drawn with line segments, rather than scaled from a bitmap and looks somewhat simplistic. However, the font can be scaled as necessary for use in programs. To set a version of the font to be 20 pixels high and 12 pixels wide, one would call:

```
var fontID : int := Font.New ("vector:20x12")
```

DOS OOT does not support any other scalable fonts. Specifically, it does not support TrueType fonts at the time of writing. DOS OOT does not support any font styles. Any call to **Font.New** will have the style ignored.

Details

If the **Font.New** call fails, then it returns 0. Also **Error.Last** will return a non-zero value indicating the reason for the failure. **Error.LastMsg** will return a string which contains the textual version of the error.

It is quite possible for **Error.Last** to be set, even if the call succeeds. **Font.New** will report success even if unable to successfully match the requested font with the available resources. A font will be set that matches as closely as possible the requested font and **Last.Error** will be set to indicate that some substitutions were required.

Example

The program prints out several phrases in a variety of fonts.

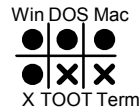
```
var font1, font2, font3, font4 : int
font1 := Font.New ("serif:12")
font2 := Font.New ("sans serif:18:bold")
font3 := Font.New ("mono:9")
font4 := Font.New ("Palatino:24:Bold,Italic")
assert font1 > 0 and font2 > 0 and font3 > 0 and font4 > 0
Font.Draw ("This is in a serif font", 50, 30, font1, red)
Font.Draw ("This is in a sans serif font", 50, 80, font2, brightblue)
Font.Draw ("This is in a mono font", 50, 130, font3, colorfg)
Font.Draw ("This is in Palatino (if available)", 50, 180, font4, green)
Font.Free (font1)
Font.Free (font2)
Font.Free (font3)
Font.Free (font4)
```

Status

Exported qualified.

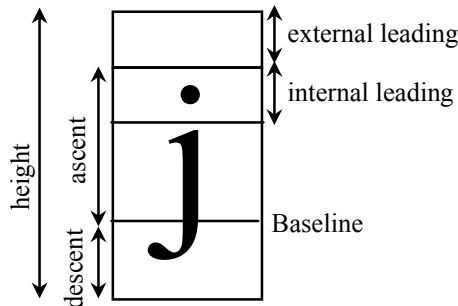
This means that you can only call the function by calling **Font.New**, not by calling **New**.

Font.Sizes



Syntax **Font.Sizes** (*fontID* : **int**, **var** *height*, *ascent*, *descent*, *internalLeading* : **int**)

Description **Font.Sizes** is used to get the metrics of a particular font. The various parts of the metric are illustrated below. Note that you can calculate the external leading by subtracting the ascent and descent from the height.



Details If **Font.Sizes** is passed an invalid font ID, a fatal error occurs. If the **Font.Sizes** call fails for other (non-fatal) reasons, the metrics for the default font will be returned. As well, **Error.Last** will return a non-zero value indicating the reason for the failure. **Error.LastMsg** will return a string which contains the textual version of the error.

Example The program gets information about 24pt Bold Italic Palatino.

```
var fontID, height, ascent, descent, internalLeading, : int
var externalLeading : int
fontID := Font.New ("Palatino:24:bold,italic")
Font.Sizes (fontID, height, ascent, descent, internalLeading)
externalLeading := height - ascent - descent
put "The height of the font is ", height, " pixels"
put "The ascent of the font is ", ascent, " pixels"
put "The descent of the font is ", descent, " pixels"
put "The internal leading of the font is ", internalLeading, " pixels"
put "The external leading of the font is ", externalLeading, " pixels"
Font.Free (fontID)
```

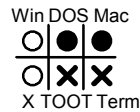
Status Exported qualified.
This means that you can only call the function by calling **Font.Sizes**, not by calling **Sizes**.

Font.StartName



Syntax	Font.StartName
Description	Font.StartName is used to start the listing of all the fonts available on the system. This procedure is called before making calls to Font.GetName to get the name of the fonts available. Once the name of a font is known, it's possible to list the available styles (using Font.GetStyle) and the available sizes (using Font.StartSize and Font.GetSize).
Example	The program lists all the fonts available on the system. <pre>var fontName : string Font.StartName loop fontName := Font.GetName exit when fontName = "" put fontName end loop</pre>
Status	Exported qualified. This means that you can only call the function by calling Font.StartName , not by calling StartName .

Font.StartSize



Syntax	Font.StartSize (<i>fontName</i> , <i>fontStyle</i> : string)
Description	Font.StartSize is used to start a listing of all the sizes for a particular font name and style. The <i>fontName</i> parameter should be an actual font name (as opposed to the default names of "serif", etc). You can get a list of the font names by using the Font.StartName and Font.GetName subprograms. The <i>fontStyle</i> parameter should be in the same format as would appear in the Font.New procedure.

Example The program lists all the fonts, styles and sizes available on the system.

```
var fontName : string
var bold, italic, underline : boolean
var size : int
var styles : array boolean, boolean, boolean of string :=
    init ("", "underline", "italic", "italic, underline", "bold", "bold,underline",
"bold,italic", "bold,italic,underline")
Font.StartName
loop
    fontName := Font.GetName
    exit when fontName = ""
    Font.GetStyle (fontName, bold, italic, underline)
    for b : false .. bold
        for i : false .. italic
            for u : false .. underline
                put fontName : 30, styles (b, i, u) : 22 ..
                Font.StartSize (fontName, styles (b, i, u) )
                loop
                    size := Font.GetSize
                    exit when size = 0
                    if size = -1 then put "scalable " ..
                    else put size, " " ..
                    end if
                end loop
                put ""
            end for
        end for
    end for
end loop
```

Status Exported qualified.
This means that you can only call the function by calling **Font.StartSize**,
not by calling **StartSize**.

Font.Width

Win	DOS	Mac
●	●	●
●	×	×
X	TOOT	Term

Syntax **Font.Width** (*txtStr* : **string**, *fontID* : **int**) : **int**

Description **Font.Width** is used to obtain the width in pixels that a specified string will
take to draw in a specified font. The *txtStr* parameter is the string. The
fontID parameter is the font in which the string would be drawn.

Details	If Font.Width is passed an invalid font ID, a fatal error occurs. If the Font.Width call fails for other (non-fatal) reasons, the width for string in the default font will be returned. As well, Error.Last will return a non-zero value indicating the reason for the failure. Error.LastMsg will return a string which contains the textual version of the error.
Example	<p>The program gets information about 24pt Bold Italic Palatino.</p> <pre> var width : int fontID := Font.New ("Palatino:24:Bold,Italic") width := Font.Width ("Test String", fontID) put "The width of \"Test String\" is ", width, " pixels" Font.Free (fontID) </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Font.Width, not by calling Width.</p>

for statement

Syntax	<p>A <i>forStatement</i> is:</p> <pre> for [decreasing] [<i>id</i>] : <i>first</i> .. <i>last</i> [by <i>increment</i>] <i>statementsAndDeclarations</i> end for </pre>
Description	<p>The statements and declarations in a for statement are repeatedly executed. In the first iteration, the identifier is assigned the value of <i>first</i>. With each additional iteration, the identifier increases by 1 (or by <i>increment</i>, if the by clause is present). The loop stops executing when adding 1 (or <i>increment</i>) to the identifier would cause the identifier to exceed <i>last</i>. <i>first</i> and <i>last</i> must be integer values (or else enumerated or char values). If you specify decreasing, then the identifier decreases by 1 (or by <i>increment</i>) each time through.</p> <p><i>Increment</i> must be a positive integer value. When the by clause is present, the for loop terminates as soon as the identifier would become greater than <i>last</i>, unless decreasing is present. If decreasing is present, the loop terminates when the identifier would become less than <i>last</i>.</p>
Details	<p>The identifier is checked before it is added to (or subtracted from). This means that the loop</p> <pre> for <i>i</i> : 1 .. maxint </pre> <p>will not cause an overflow.</p>

Example Output 1, 2, 3 to 10.

```
for i : 1 .. 10
  put i
end for
```

Example Output 1, 3, 5, 7 and 9.

```
for i : 1 .. 10 by 2
  put i
end for
```

Example Output 10, 9, 8, down to 1.

```
for decreasing j : 10 .. 1
  put j
end for
```

Example Output 10, 6, and 2.

```
for decreasing j : 10 .. 1 by 4
  put j
end for
```

Example Output 1.

```
for j : 1 .. 10 by 20
  put j
end for
```

Example Output nothing.

```
for j : 5 .. 2
  put j
end for
```

Details The **for** statement declares the counting identifier (a separate declaration should not be given for *i* or *j*). The scope of this identifier is restricted to the **for** statement.

If *first* is a value beyond *last*, there will be no repetitions (and no error message). The counting identifier is always increased (or decreased) by 1 or *increment* if the **by** clause is present. Executing an **exit** statement inside a **for** statement causes a jump to just beyond **end for**. You are not allowed to change the counting variable (for example, you are not allowed to write *i* := 10).

The counting identifier can be omitted. In this case, the statement is just as before, except that the program cannot use the value of the identifier.

If **decreasing** is not present, *first* .. *last* can be replaced by the name of a subrange type, for example by *dozen*, declared by:

```
type dozen : 1..12
```

Procedures, functions and modules cannot be declared inside a **for** statement. Just preceding the statements and declarations, you are allowed to write an "invariant clause" of the form:

```
invariant trueFalseExpn
```

This clause is equivalent to: **assert** *trueFalseExpn*.

fork statement

Dirty parts

Syntax	<p>A <i>forkStatement</i> is:</p> $\mathbf{fork} \text{ } processId \left[\left(\left[\textit{expn} \left\{ , \textit{expn} \right\} \right] \right) \right] \left[: \textit{reference} \left[, \textit{expn} \left[, \textit{reference} \right] \right] \right]$
Description	<p>A fork activates (starts the concurrent execution of) a process declaration. If the process has parameters, a parenthesized list of expressions (<i>expns</i>) must follow the process' name (<i>processId</i>).</p>
Example	<p>This program initiates (forks) two concurrent processes, one of which repeatedly outputs Hi and the other Ho. The resulting output is an unpredictable sequence of Hi's and Ho's, as <i>greetings</i> executes twice concurrently, one instance with its <i>word</i> set to Hi and the other with its <i>word</i> set to Ho.</p> <pre>process greetings (word : string) loop put word end loop end greetings fork greetings ("Hi") fork greetings ("Ho")</pre>
Details	<p>See procedure declaration for details about parameters. The first optional <i>reference</i> in the fork statement must be a boolean variable reference. The fork sets this to true if the process is actually activated. If this fails to occur (probably because stack space could not be allocated), this <i>reference</i> is set to false. If the fork fails but this reference is omitted, an exception occurs. See exception handlers.</p> <p>The optional <i>expn</i> specifies the number of bytes for the process' stack; this overrides the optionally given stack size in the process declaration. The second optional <i>reference</i> must be a variable reference with the type addressint. See addressint. This variable is set to identify the process activation. This reference has the implementation-dependent meaning of locating the process' internal descriptor.</p> <p>In this explanation of the fork statement, we have up to this point ignored the possibility of processes exported from modules. If the process is being forked from outside of a module from which it has been exported, the syntax of the fork statement is:</p>

fork moduleId . procedureId [(expn { , expn })] ...

In other words, the module's name and a dot must precede the process' name.

forward subprogram declaration

Syntax A *forwardDeclaration* is:

forward *subprogramHeader*
[**import** *importItem* { , *importItem* }]

Description A procedure or function is declared to be **forward** when you want to define its header but not its body. This is the case when one procedure or function calls another, which in turn calls the first; this situation is called *mutual recursion*. The use of **forward** is necessary in this case, because every item must be declared before it can be used.

Example This example program evaluates an input expression *e* of the form *t* { + *t* } where *t* is of the form *p* { * *p* } and *p* is of the form (*e*) or an explicit real expression. For example, the value of 1.5 + 3.0 * (0.5 + 1.5) halt is 7.5.

```

var token : string

forward procedure expn ( var eValue : real )

forward procedure term ( var tValue : real )

forward procedure primary ( var pValue: real )

body procedure expn
  var nextValue : real
  term ( eValue )           % Evaluate t
  loop                     % Evaluate { + t}
    exit when token not= "+"
    get token
    term ( nextValue )
    eValue := eValue + nextValue
  end loop
end expn

body procedure term
  var nextValue : real
  primary ( tValue )       % Evaluate p
  loop                     % Evaluate { * p}
    exit when token not= "*"
    get token

```

```

        primary ( nextValue )
        tValue := tValue + nextValue
    end loop
end term

body procedure primary
    if token = "(" then
        get token
        expn ( pValue )          % Evaluate (e)
        assert token = ")"
    else
        % Evaluate "explicit real"
        pValue := strreal ( token )
    end if
    get token
end primary

get token          % Start by reading first token
var answer : real
expn ( answer )    % Scan and evaluate input expression
put "Answer is ", answer

```

- Details Following a **forward** procedure or function declaration, the **body** of the procedure must be given at the same level (in the same sequence of statements and declarations as the **forward** declaration). This is the only use of the keyword **body**. See also **body**.
- Any procedure or function that is declared using **forward** requires an **import** list. In this list, imported procedures or functions that have not yet appeared must be listed as **forward**. For example, the import list for *expn* is **import forward term ...** Before a procedure or function can be called, before its body appears, and before it can be passed as a parameter, its header as well as headers of procedures or functions imported directly or indirectly by it must have appeared.
- The keyword **forward** is also used in **collection** and **type** declarations.
- See also **collections** and **type** declarations.

frealstr real-to-string function

- Syntax **frealstr (*r* : real, *width*, *fractionWidth* : int) : string**
- Description The **frealstr** function is used to convert a real number to a string. For example, **frealstr** (2.5e1, 5, 1)="b25.0" where *b* represents a blank. The string is an approximation to *r*, padded on the left with blanks as necessary to a length of *width*.

The number of digits of the fraction to be displayed is given by *fractionWidth*.

The *width* must be non-negative. If the *width* parameter is not large enough to represent the value of *r*, it is implicitly increased as needed.

The *fractionWidth* must be non-negative. The displayed value is rounded to the nearest decimal equivalent with this accuracy. In the case of a tie, the value is rounded to the next larger value. The result string is of the form:

{blank} [-]{digit}. {digit}

If the leftmost digit is zero, then it is the only digit to the left of the decimal point.

The **frealstr** function approximates the inverse of **strreal**, although round-off errors keep these from being exact inverses.

See also the **erealstr**, **realstr**, **strreal**, **intstr** and **strint** functions.

free statement

Syntax A *freeStatement* is:

free [*collectionOrClassId*,]
pointerVariableReference

Description A **free** statement destroys (deallocates) an element that has been allocated by a **new** statement.

Example Using a collection, declare a list of records and allocate one of these records. Then deallocate the record.

```
var list : collection of  
    record  
        contents : string ( 10 )  
        next : pointer to list  
                % Short form: next : ^ list  
    end record  
var first : pointer to list      % Short form: var next : ^ list  
new list, first  
    % Allocate an element of list; place its location in first  
    % Short form: new first  
...  
free list, first % Deallocate the element of list located by first  
                % Short form: free first
```


Details	<p>The free statement sets the pointer variable to the nil value. See the new statement for examples of allocating elements of classes and values of types. If the pointer locates a type, the <i>collectionOrClassId</i> in the free statement must be omitted.</p> <p>An imported class can have one of its objects destroyed (by the free statement) only if the class is imported var.</p> <p>The <i>collectionOrClassId</i> is optional in the free statement.</p>
See also	class and collection declarations, the pointer type, the new statement and the nil value.

function declaration

Syntax	<p>A <i>functionDeclaration</i> is:</p> <pre> function <i>id</i> [([<i>paramDeclaration</i> { , <i>paramDeclaration</i> }])] : <i>typeSpec</i> <i>statementsAndDeclarations</i> end <i>id</i> </pre>
Description	<p>A function declaration creates (but does not run) a new function. The name of the function (<i>id</i>) is given in two places, just after function and just after end.</p>
Example	<pre> function <i>doubleIt</i> (<i>x</i> : real) : real result 2.0 * <i>x</i> end <i>doubleIt</i> put <i>doubleIt</i> (5.3) % This outputs 10.6 </pre>
Details	<p>The set of parameters declared with the function are called <i>formal</i> parameters. For example, in the <i>doubleIt</i> function, <i>x</i> is a formal parameter. A function is called (invoked) by a <i>function call</i> which consists of the function's name followed by the parenthesized list of <i>actual</i> parameters (if any). For example, <i>doubleIt</i> (5.3) is a call having 5.3 as an actual parameter. If there are no parameters and no parentheses, the call does not have parentheses. The keyword function can be abbreviated to fcn. See also <i>functionCall</i> and <i>procedureDeclaration</i>.</p> <p>Each actual non-var parameter must be assignable to the type of its corresponding formal parameter. See also <i>assignability</i>.</p>

A function must finish by executing a **result** statement, which produces the function's value. In the above example, the **result** statement computes and returns the value $2.0 * x$.

In principle, a function (1) should not change any variables outside of itself (global variables) or (2) should not have **var** parameters. In other words, it should have no *side effects*. The original implementation prevented (1) and (2) and thereby prevented function side effects. Current implementations of Turing do not enforce this restriction.

The upper bounds of arrays and strings that are parameters may be declared to be an asterisk (*), meaning the bound is that of the actual parameter. See *paramDeclaration* for details about parameters.

Procedures and functions cannot be declared inside other procedures and functions.

The syntax of a *functionDeclaration* presented above has been simplified by leaving out the optional result identifier, **import** list, **pre** condition, **init** clause, **post** condition and exception handler. The full syntax is

```
function [ pervasive ] id
  [ ( [ paramDeclaration { paramDeclaration } ] ) ]
  [ resultId ] : typeSpec
  [ pre trueFalseExpn ]
  [ init id := expn { , id := expn } ]
  [ post trueFalseExpn ]
  [ exceptionHandler ]
  statementsAndDeclarations
end id
```

See also

import list, **pre** condition, **init** clause, **post** condition and *exceptionHandler* for explanations of these additional features.

See also **pervasive**. The *resultId* is the name of the result of the function and can be used only in the **post** condition.

A function must be declared before being called; to allow for mutually recursive procedures and functions, there are **forward** declarations with later declaration of the procedure or function **body**. See **forward** and **body** declarations for explanations.

You declare parameterless functions using an empty parameter list. When this is done, a call to the function must include an empty parameter list.

functionCall

Syntax

A *functionCall* is:

```
functionId [ ( [ expn { , expn } ] ) ]
```

Description	A function call is an expression that calls (invokes or activates) a function . If the function has parameters, a parenthesized list of expressions (<i>expns</i>) must follow the function's name (<i>functionId</i>).
Example	<p>This function takes a string containing a blank and returns the first word in the string (all the characters up to the first blank).</p> <pre> function <i>firstWord</i> (<i>str</i> : string) : string for <i>i</i> : 1 .. length (<i>str</i>) if <i>str</i> (<i>i</i>) = " " then result <i>str</i> (1 .. <i>i</i> - 1) end if end for end <i>firstWord</i> put "The first word is: ", <i>firstWord</i> ("Henry Hudson") % The output is Henry. </pre>
Details	<p>The parameter declared in the header of a function, is a <i>formal</i> parameter, for example, <i>str</i> above is a formal parameter. Each expression in the call is an <i>actual</i> parameter, for example, <i>sample</i> above is an actual parameter.</p> <p>Each actual parameter passed to its non-var formal parameter must be assignable to that parameter (see <i>assignability</i> for details). See also <i>functionDeclaration</i> and <i>procedureDeclaration</i>.</p> <p>In this explanation of <i>functionCall</i>, we have up to this point ignored the possibility of functions exported from modules. If the function is being called from outside of a module from which it has been exported, the syntax of the <i>functionCall</i> is:</p> $moduleId . functionId [(expn \{, expn \})]$ <p>In other words, the module or monitor name and a dot must precede the function's name. If the function is being called from outside of a class from which it has been exported, the syntax of the <i>functionCall</i> is one of:</p> <p>(a) $classId (p) . functionId [([expn \{, expn \}])]$</p> <p>(b) $p \rightarrow functionId [([expn \{, expn \}])]$</p> <p>In these <i>p</i> must be a pointer value that locates an object in the class. Form (b) is a short form for form (a).</p>
See also	class .

get file statement

Syntax A *getStatement* is:

get [: streamNumber ,] getItem { , getItem }

Description The **get** statement inputs each of the *getItem*s. Ordinarily, the output comes from the keyboard. However, if the *streamNumber* is present, the input comes from the file specified by the stream number (see the **open** statement for details). Also, input can be redirected so it is taken from a file rather than the keyboard. Check the documentation on the environment for instructions on doing so.

The syntax of a *getItem* is one of:

- (a) *variableReference*
- (b) **skip**
- (c) *variableReference* : *
- (d) *variableReference* : *widthExpn*

These items are used to support three kinds of input:

- (1) token and character oriented input: supported by forms (a) and (b),
- (2) line oriented input: supported by form (c), and
- (3) character oriented input: supported by form (d).

Examples of these will be given, followed by detailed explanations.

Example Token-oriented input.

```
var name, title : string
var weight : real
get name      % If input is Alice, it is input into name
get title     % If input is "A lady", A lady is input
               var weight      % If input is 9.62, it is input into weight
```

Example Line-oriented input.

```
var query : string
get query : * % Entire line is input into query
```

Example Character-oriented input.

```
var code : string
get code : 2      % Next 2 characters are input into code.
```

Details A *token* is a sequence of characters surrounded by *white space*, where *white space* is defined as the characters: blank, tab, form feed, new line, and carriage return as well as end-of-file. The sequence of characters making up the token are either all non-white space or else the token must be a quoted string (an explicit string constant). When the *variableReference* in form (a) is a string, integer, real, **intn**, **natn**, or **realn**. Turing skips white space, reads a token into the *variableReference*, and then skips white space (stopping at the beginning of the next line).

If the *variableReference* is a string, the token is assigned to the variable (if the token is quoted, the quotation marks are first removed). See the examples involving *name* and *title* above. If the *variableReference* is an integer or a real, the token is converted to be numeric before being assigned to the variable. See the example involving *weight* above.

When the input is coming from the keyboard, no input is done until Return is typed. The line that is input may contain more than one token. Any tokens that are not input by one **get** statement will remain to be input by the next **get** statement.

Turing has been modified so that token-oriented input now also skips white space following the token, but does not skip beyond the beginning of the next line. This change implies that form (b) is usually not needed, as **skip** was used to skip white space after the token.

Form (a) supports **char** and **char(n)**. If the type is **char**, exactly one character is read, with no skipping of white space before or after. This character may be, for example, a blank or a carriage return. If the type is **char(n)**, exactly *n* characters are read, with no skipping of white space.

Example Inputting **char** and **char(n)** types using form (a). The statement *get c:1* is not legal, because length specification is not allowed with character variables.

```
var c : char
var d : char ( 3 )
get c           % Read one character.
get d           % Read three characters
```

Details Form (a) supports enumerated types. If the type is an enumerated type, then the token read in must be one of the elements of the enumerated type.

Example Inputting an enumerated type using form (a). The statement *get c:1* is not legal, because length specification is not allowed with enumerated variables.

```
type colors : enum (red, blue, green)
var c : colors
get c           % Read one of red, green or blue
```

Details Form (a) supports **boolean**. If the type is an **boolean** type, then the token read in must be one of "true" or "false"

Example Inputting a **boolean** type using form (a). The statement *get c:1* is not legal, because length specification is not allowed with **boolean** variable.

```
var tf : boolean
get tf          % Read one of true or false
```

Details In form (b) of *getItem*, **skip** causes white space in the input to be skipped until non-white space (a token) or the end-of-file is reached. This is used when the program needs to determine if there are more tokens to be input. To determine if there are more tokens to be read, the program should first **skip** over any possible white space (such as a final new line character) and then test to see if **eof** (end-of-file) is true. This is illustrated in this example:

Example Using token-oriented input, input and then output all tokens. This example gives what used to be the standard way of reading tokens up to end of file. With the new meaning of form (a) for reading tokens, the **get skip** line can be omitted. This omission is possible because the line **get word** now automatically skips white space following the input value, up to the beginning of the next line.

```

var word : string
loop
  get skip      % Skip over any white space
  exit when eof % Are there more characters?
  get word      % Input next token
  put word      % Output the token
end loop

```

In the above and the next example, if the input has been redirected so that it is from a file, **eof** becomes true exactly when there are no more characters to be read. If the input is coming from the keyboard, you can signal **eof** by typing control-Z (on a PC) or control-D (on UNIX).

Details In form (c) of *getItem*, the *variableReference* is followed by ***** which implies line-oriented input. This form causes the entire line (or the remainder of the current line) to be read. In this case the variable must be a string (not an integer or real). The new line character at the end of the line is discarded. It is an error to try to read another line when you are already at the end of the file. The following example shows how to use line-oriented input to read all lines in the input.

Example Using line-oriented input, input and then output all lines.

```

var line : string
loop
  exit when eof % Are there more characters?
  get line : * % Read entire line
  put line
end loop

```

Details In form (d) of *getItem*, the *variableReference* is followed by

: widthExpn which specifies character-oriented input. This form causes the specified number (*widthExpn*) of characters to be input (or all of the remaining characters if not enough are left). If no characters remain, the null string is read and no warning is given. In this form, the new line character is actually input into the *variableReference* (this differs from line-oriented input which discards new line characters). The following example shows how to use character-oriented input to read each character of the input. Form (d) can be used with **string** and **char**(*n*) variables, but not with **char**, **int** or any other type.

Example Using character-oriented input, input and then output all characters.

```

var ch : string ( 1 )
loop
  exit when eof % Are there more characters?
  get ch : 1 % Read one character
  put ch .. % Output the character, which
            % may be a new line character
end loop

```

Example Using character-oriented input, input two characters.

```

var d : char ( 3 ) := 'abc'

```

get *d* : 2 % Read two character (replace 'ab')

See also **read** statement, which provides binary file input.

getch get character procedure

Syntax **getch** (**var** *ch* : **string** (1))

Description The **getch** procedure is used to input a single character without waiting for the end of a line. The parameter *ch* is set to the next character in the keyboard buffer (the oldest not-yet-read character).

Example This program contains a procedure called *getKey* which causes the program to wait until a key is pressed.

```
setscreen ("graphics")

procedure getKey
  var ch : string (1)
  getch (ch)
end getKey

for i : 1 .. 1000
  put i : 4, " Pause till a key is pressed"
  getKey
end for
```

Details The screen should be in a "screen" or "graphics" mode. See the **setscreen** procedure for details. If the screen is not in one of these modes, it will automatically be set to "screen" mode.

On IBM PC's some keys, such as the left arrow key, insert key, delete key, and function keys do not produce ordinary character values. These keystrokes are returned by **getch** as their "scan code" with 128 added to them, unless the scan code already has a value of 128 or greater. This provides a unique value for every key on the keyboard. See Appendix D for these codes.

See also **hasch** (has character) procedure which is used to see if a character has been typed but not yet read.

See also predefined unit **Input**.

getchar get character function

Syntax **getchar : char**

Description The **getchar** function is used to input a single character without waiting for the end of a line. The next character in the keyboard buffer (the oldest not-yet-read character) is returned.

Example This program contains a procedure called *getKey* which causes the program to wait until a key is pressed.

```
setscreen ("graphics")

procedure getKey
  var ch : char
  ch := getchar
end getKey

for i : 1 .. 1000
  put i : 4, " Pause till a key is pressed"
  getKey
end for
```

Details The screen should be in a "screen" or "graphics" mode. See the **setscreen** procedure for details. If the screen is not in one of these modes, it will automatically be set to "screen" mode.

On IBM PC's some keys, such as the left arrow key, insert key, delete key, and function keys do not produce ordinary character values. These keystrokes are returned by **getchar** as their "scan code" with 128 added to it, unless the scan code already has a value of 128 or greater. This provides a unique value for every key on the keyboard. See Appendix D for these codes.

See also **hasch** (has character) procedure which is used to see if a character has been typed but not yet read.

See also predefined unit **Input**.

getenv get environment function

Win DOS Mac
● ● X
● ● X
X TOOT Term

Syntax	getenv (<i>symbol</i> : string) : string
Description	The getenv function is used to access the environment string whose name is <i>symbol</i> . These strings are determined by the shell (command processor) or the program that caused your program to run. See also the nargs and fetcharg functions.
Example	<p>On a PC, this retrieves the environment variable USERLEVEL and prints extra instructions if USERLEVEL had been set to NOVICE. USERLEVEL can be set to NOVICE with the command SET USERLEVEL = NOVICE in the autoexec.bat file or in any batch file.</p> <pre>const userLevel : string userLevel := getenv ("USERLEVEL") if userLevel = "NOVICE" then ... % put a set of instructions end if</pre>
See also	See also predefined unit Sys .

getpid get process id function

Win DOS Mac
X X X
● ● X
X TOOT Term

Syntax	getpid : int
Description	<p>The getpid function is used to determine the I.D. (number) that identifies the current operating system task (process). Beware that there are processes, activated by the fork statement, that are independent of the operating systems tasks.</p> <p>Under UNIX, the number is used, for example, for creating a unique name of a file.</p>
See also	nargs , fetcharg and getenv . See also predefined unit Sys .

getpriority function

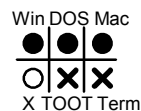
Syntax	getpriority : nat
Description	The getpriority function returns the priority of an executing process in a concurrent program. A smaller value means a faster speed.
See also	setpriority , fork and monitor . See also predefined unit Concurrency .

GUI



Description	<p>This unit contains the predefined subprograms for creating and using a GUI (Graphical User Interface). Elements of the GUI include buttons, check boxes, text boxes, scroll bars, menus, etc.</p> <p>For a general introduction to the the GUI module, see Chapter 5 of this book.</p> <p>All routines in the GUI module are exported qualified (and thus must be prefaced with "GUI.").</p>
-------------	--

GUI.AddLine



Syntax	GUI.AddLine (<i>widgetID</i> : int , <i>text</i> : string)
--------	---

Description	GUI.AddLine adds text and a newline to the current line of the text box specified by <i>widgetID</i> . It is essentially equivalent to put text in the text box. GUI.AddLine scrolls the text box (if necessary) so that the added text is now visible. The <i>widgetID</i> parameter must be the widget id of a text box. The <i>text</i> parameter is the text to be added to the text box.
Example	The following creates a text box and puts the numbers from 1 to 25 in it. <pre> import GUI in "%oot/lib/GUI" var boxID : int := GUI.CreateTextBox (50, 50, 200, 200) for i : 1 .. 25 GUI.AddLine (boxID, intstr (i)) end for loop exit when GUI.ProcessEvent end loop </pre>
Status	Exported qualified. This means that you can only call the function by calling GUI.AddLine , not by calling AddLine .
See also	GUI.CreateTextBox .

GUI.AddText



Syntax	GUI.AddText (<i>widgetID</i> : int , <i>text</i> : string)
Description	GUI.AddText adds text to the current line of the text box specified by <i>widgetID</i> . It does not add a newline after the text. It is essentially equivalent to put text ... in the text box. GUI.AddLine scrolls the text box (if necessary) so that the added text is now visible. The <i>widgetID</i> parameter must be the widget id of a text box. The <i>text</i> parameter is the text to be added to the text box.
Details	To force a text box to scroll to the end of the text without adding any extra text, call GUI.AddText with "" (the null string) for the <i>text</i> parameter.
Example	The following creates a text box and puts the numbers from 1 to 26 followed by the appropriate letter of the alphabet in it. <pre> import GUI in "%oot/lib/GUI" var boxID : int := GUI.CreateTextBox (50, 50, 200, 200) for i : 1 .. 26 GUI.AddText (boxID, intstr (i)) GUI.AddText (boxID, " ") </pre>

	<pre> GUI.AddLine (boxID, chr (64 + i)) end for loop exit when GUI.ProcessEvent end loop </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.AddText, not by calling AddText.</p>
See also	GUI.CreateTextBox.

GUI.Alert[2,3,Full]



Syntax	<pre> GUI.Alert (<i>title</i>, <i>msg</i> : string) GUI.Alert2 (<i>title</i>, <i>msg1</i>, <i>msg2</i> : string) GUI.Alert3 (<i>title</i>, <i>msg1</i>, <i>msg2</i>, <i>msg3</i> : string) GUI.AlertFull (<i>title</i> : string, <i>msg</i> : array 1 .. * of string, <i>button</i> : string) </pre>
Description	<p>Displays a dialog box with the string specified by <i>msg</i> in it. There is a single button labelled <i>OK</i> which dismisses the dialog and resumes execution. The <i>title</i> parameter specifies the window title under Microsoft Windows. On the Apple Macintosh, there is no title, so do not assume the user will see the title. The dialog box is centered on the screen.</p> <p>The GUI.Alert2 and GUI.Alert3 procedures allow the user to specify a two or three line message respectively. The GUI.AlertFull procedure allows the user to specify any number of lines of text in the string array specified by <i>msg</i> as well as the text in the dismissal button. Any empty strings at the end of the array are not displayed.</p> <p>Note: This function is not available in the current version of the GUI Procedure Library (shipping with WinOOT 3.1, DOS OOT 2.5 and MacOOT 1.5). It is documented here for use with future shipping version of Object Oriented Turing. It is likely to be implemented in the version of Object Oriented Turing released in September 2000. Check the release notes that are found in the on-line help to find out if this function is now available.</p>

Example The following program asks the user for the name of a file puts up an alert dialog box if it fails.

```
import GUI in "%oot/lib/GUI"

var fileName : string
var streamNumber : int

loop
  fileName := GUI.SaveFile ("Save As")
  open : streamNumber, fileName, put
  exit when streamNumber > 0
  GUI.Alert ("Open Failure", "\" + fileName +
    "\" could not be opened")
end loop
```

Example The following program asks the user for the name of a file puts up a more complete alert dialog box if it fails.

```
import GUI in "%oot/lib/GUI"

var fileName : string
var streamNumber : int
loop
  fileName := GUI.SaveFile ("Save As")
  open : streamNumber, fileName, put
  exit when streamNumber > 0
  GUI.Alert2 ("Open Failure",
    "\" + fileName + "\" could not be opened.",
    "Reason: " + Error.LastMsg)
end loop
```

Example The following program fragment displays an alert with four lines of text and a button that says "Abort".

```
var message : array 1 .. 10 of string
for i : 1 .. 10
  message (i) := ""
end for
...
message (1) := "The program must now quit"
message (2) := "because of an unrecoverable error."
message (3) := "A Read Error occurred while reading"
message (4) := "file \" + fileName + "\" ."
message (5) := Error.LastMsg
GUI.AlertFull ("Error", message, "Abort")
```

Status Exported qualified.

This means that you can only call the function by calling **GUI.Alert**, not by calling **Alert**.



GUI.Choose[Full]

Syntax	<pre> GUI.Choose (<i>title</i>, <i>msg1</i>, <i>msg2</i>, <i>msg3</i> : string, <i>btn1</i>, <i>btn2</i>, <i>btn3</i> : string) : int GUI.ChooseFull (<i>title</i> : string, <i>msg</i> : array 1 .. * of string, <i>btn1</i>, <i>btn2</i>, <i>btn3</i> : string, <i>defaultBtn</i> : int) : int </pre>
Description	<p>Displays a dialog box with text and from one to three buttons. The user selects a button to dismiss the dialog. The number of the button pressed is returned by the function. The dialog box is centered on the screen.</p> <p>The <i>title</i> parameter specifies the title in the window bar of the dialog box. The Apple Macintosh does not have a title bar, so do not assume that the user will see the string in the <i>title</i> parameter. The message is specified by strings in <i>msg1</i>, <i>msg2</i> and <i>msg3</i> for GUI.Choose and the string array <i>message</i> for GUI.ChooseFull. In each case, empty strings at the end of the list of strings are ignored. The <i>btn1</i>, <i>btn2</i>, and <i>btn3</i> parameters specify the text to appear in the buttons. If the text is an empty string (""), the button is not displayed.</p> <p>The function returns the button number from one to three that was chosen.</p> <p>The <i>defaultBtn</i> parameter in GUI.ChooseFull specifies which, if any, button should be the default button. The default button is selected if the user presses Enter. If the default button is 0, then no button is highlighted as the default button.</p> <p>Note: This function is not available in the current version of the GUI Procedure Library (shipping with WinOOT 3.1, DOS OOT 2.5 and MacOOT 1.5). It is documented here for use with future shipping version of Object Oriented Turing. It is likely to be implemented in the version of Object Oriented Turing released in September 2000. Check the release notes that are found in the on-line help to find out if this function is now available.</p>
Example	<p>The following program asks if the user wants coffee or tea and set <i>wantsCoffee</i> appropriately.</p> <pre> import GUI in "%oot/lib/GUI" var <i>wantsCoffee</i> : boolean var <i>choice</i> : int := GUI.Choose ("Beverage Choice", "Do you want coffee or tea?", "", "", "Coffee", "Tea", "") if <i>choice</i> = 1 then <i>wantsCoffee</i> := true else <i>wantsCoffee</i> := false end if </pre>

Example The following program asks the user whether they want to save their work, don't save their work or Cancel.

```
import GUI in "%oot/lib/GUI"

% Returns false if cancelling operation
procedure CheckUnsavedWork : boolean
  var message : array 1 .. 3 of string
  message (1) := "Changes to " + fileName + " have not been "
  message (2) := "saved. Unsaved work will be lost. Do you "
  message (3) := "want to save before quitting."
  var choice : int := GUI.ChooseFull ("Save Before Quit",
    message, "Save", "Don't Save", "Cancel", 1)
  if choice = 1 then
    SaveWork
  elsif choice = 3 then
    return false
  end if
  return true
end CheckUnsavedWork
```

Status Exported qualified.
This means that you can only call the function by calling **GUI.Choose**, not by calling **Choose**.

GUI.ClearText



Syntax **GUI.ClearText** (*widgetID* : **int**)

Description Clears all the text in a text box specified by *widgetID*. The *widgetID* parameter must be the widget id of a text box.

Example The program lists 25 numbers in a text box. Every time the button is pressed, it clears the text box and prints the next 25 numbers.

```
import GUI in "%oot/lib/GUI"
var boxID, buttonID, start : int
start := 1

procedure PrintTwentyFive
  GUI.ClearText (boxID)
  for i : start .. start + 24
    GUI.AddLine (boxID, intstr (i))
  end for
  start += 25
end PrintTwentyFive

boxID := GUI.CreateTextBox (50, 50, 200, 200)
```

```

buttonID := GUI.CreateButton (50, 5, 0, "Next 25", PrintTwentyFive)
PrintTwentyFive
loop
  exit when GUI.ProcessEvent
end loop

```

Status Exported qualified.
 This means that you can only call the function by calling **GUI.ClearText**,
 not by calling **ClearText**.

See also **GUI.CreateTextBox**.

GUI.CloseWindow



Syntax **GUI.CloseWindow (window : int)**

Description Closes a window with widgets in it. This procedure automatically disposes
 of any widgets in the window and makes certain that the GUI Library
 recognizes that the window no longer exists. This procedure will call
 Window.Close, so there is no need for the user to do so.

Example The program opens up a window with two buttons. If the button labelled
 "Close and Open" is pressed, the window is closed and a new window with
 two buttons is opened in a random location on the screen.

```

import GUI in "%oot/lib/GUI"

const screenWidth : int := Config.Display (cdScreenWidth)
const screenHeight : int := Config.Display (cdScreenHeight)
const titleBarHeight : int := 32
const windowEdgeSize : int := 13
const windowWidth : int := 150
const windowHeight : int := 100
var windowID, windowNumber, closeButton, quitButton : int := 0

procedure CloseAndOpen
  if windowID not= 0 then
    GUI.CloseWindow (windowID)
  end if
  windowNumber += 1
  var xPos : int := Rand.Int (0, screenWidth - windowWidth -
    windowEdgeSize)
  var yPos : int := Rand.Int (0, screenHeight - windowHeight -
    titleBarHeight)
  windowID := Window.Open ("title:Window #" +
    intstr (windowNumber) + ",graphics:" +
    intstr (windowWidth) + "," + intstr (windowHeight) +

```



```

        "position:" + intstr (xPos) + ";" + intstr (yPos))
closeButton := GUI.CreateButton (10, 60, 130,
    "Close And Open", CloseAndOpen)
quitButton := GUI.CreateButton (10, 10, 130, "Quit", GUI.Quit)
end CloseAndOpen

CloseAndOpen
loop
    exit when GUI.ProcessEvent
end loop

    GUI.CloseWindow (windowID)

```

Status

Exported qualified.

This means that you can only call the function by calling **GUI.CloseWindow**, not by calling **CloseWindow**.

GUI.CreateButton[Full]



Syntax

```

GUI.CreateButton ( x, y, width : int, text : string,
    actionProc : procedure x () ) : int

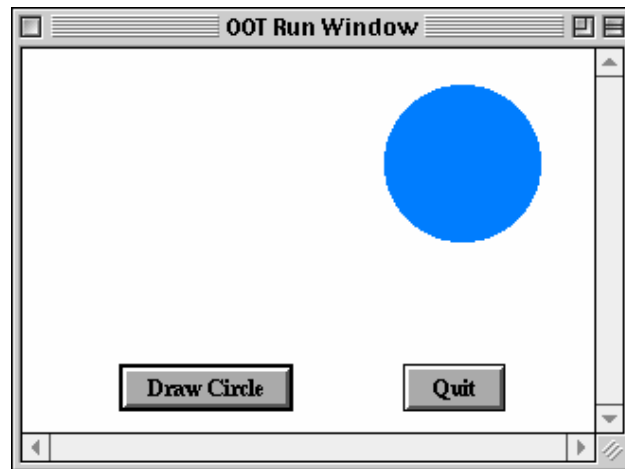
GUI.CreateButtonFull ( x, y, width : int, text : string,
    actionProc : procedure x (), height : int,
    shortcut : char, default : boolean ) : int

```

Description

Creates a button and returns the button's widget ID.

The button widget is used to implement a textual button. When you click on a button, the button's *action procedure* is called. If a button is given a short cut, then entering the keystroke will cause the *action procedure* to be called. It will not visibly cause the button to depress.



Two Buttons

The *x* and *y* parameters specify the lower-left corner of the button. The *width* parameter specifies the width of the button. If *width* is less than the size necessary to display the button, the button is automatically enlarged to fit the text. The *text* parameter specifies the text to appear in the button. The *actionProc* parameter is the name of a procedure that is called when the button is pressed.

For **GUI.CreateButtonFull**, the *height* parameter specifies the height of the button. If *height* is less than the size necessary to display the button, the button is automatically enlarged to fit the text. The *shortcut* parameter is the keystroke to be used as the button's shortcut. The *default* parameter is a boolean indicating whether the button should be the default button. If there is already a default button, and *default* is set to true, then this button becomes the new default button.

Example The following program creates two buttons, one which draws a random circle on the screen and one which quits the program.

```
import GUI in "%oot/lib/GUI"

procedure DrawRandomCircle
  var r : int := Rand.Int (20, 50)
  var x : int := Rand.Int (r, maxx - r)
  var y : int := Rand.Int (r, maxy - r)
  var c : int := Rand.Int (0, maxcolor)
  Draw.FillOval (x, y, r, r, c)
  % In case we drew over the buttons, redraw them.
  GUI.Refresh
end DrawRandomCircle

View.Set ("graphics:300;200")
var draw : int := GUI.CreateButtonFull (50, 10, 0, "Draw Circle",
  DrawRandomCircle, 0, '^D', true)
var quitBtn : int := GUI.CreateButton (200, 10, 0, "Quit", GUI.Quit)
loop
```

**exit when GUI.ProcessEvent
end loop**

Details	<p>When GUI.CreateButton or GUI.CreateButtonFull is called, the newly created button will be displayed immediately unless GUI.DisplayWhenCreated has been called with the <i>display</i> parameter set to false.</p> <p>If a button's width or height is set to zero (or not specified at all), then the button is shaped to fit the text.</p> <p>A button can be the default button for a window. The default button is drawn with a thicker border around it. If the user presses ENTER in a window with a default button, the default button's <i>action procedure</i> is called.</p> <p>When a button is not enabled, the text in the button is grayed out and the button no longer responds to any mouse clicks or keystrokes until the button is enabled again.</p>
Details	<p>The following GUI subprograms can be called with a button as the <i>widgetID</i> parameter:</p> <p style="text-align: center;">GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.SetLabel, GUI.SetDefault</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateButton, not by calling CreateButton.</p>
See also	<p>GUI.SetLabel for changing the button's text and GUI.SetDefault for setting the default button in a window.</p>

GUI.CreateCanvas[Full]



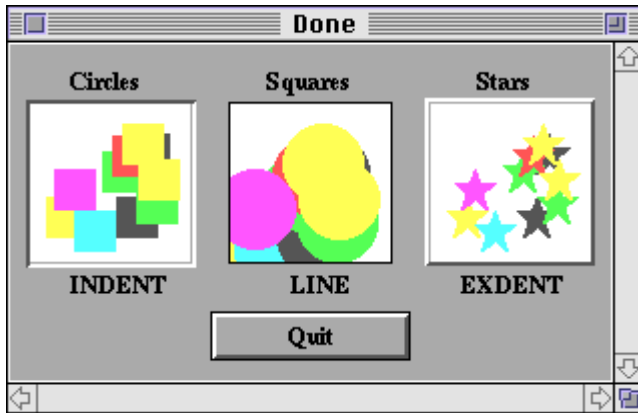
Syntax	<p>GUI.CreateCanvas (<i>x, y, width, height</i> : int) : int</p> <p>GUI.CreateCanvasFull (<i>x, y, width, height</i> : int, <i>border</i> : int, <i>mouseDown</i> : procedure <i>x</i> (<i>mx, my</i> : int), <i>mouseDrag</i> : procedure <i>x</i> (<i>mx, my</i> : int), <i>mouseUp</i> : procedure <i>x</i> (<i>mx, my</i> : int)) : int</p>
--------	--

Description Creates a canvas and returns the canvas' widget ID.

A canvas is a drawing surface for use by the program. It differs from just using the window surface to draw on in that (0, 0) represents the lower-left corner of the canvas and all drawing is clipped to the canvas. (This means that if you accidentally attempt to draw outside of the canvas, it will not actually draw beyond the border of the canvas.)

Canvases have procedures that emulate all the procedures in the Draw module as well as a procedure to emulate Font.Draw, Pic.Draw, Pic.New, Pic.ScreenLoad and Pic.ScreenSave.

You can get mouse feedback from a canvas. Using the *CreateCanvasFull* method, you can specify three routines that are called when the mouse button is depressed while pointing in a canvas. One routine will be called when the user presses the mouse button down in a canvas. Another routine will be called while the user drags the mouse with the mouse button down. This routine is repeatedly called whenever the mouse changes position while the mouse button is down. The last routine is called when the mouse button is released. All three routines take an *x* and *y* parameter, which is the location of the mouse with respect to the canvas (i.e. (0, 0) is the lower-left corner of the canvas).



Output of Canvases.dem

The *x* and *y* parameters specify the lower-left corner of the canvas. The *width* and *height* parameters specify the width and height of the canvas.

For **GUI.CreateCanvasFull**, the *border* parameter specifies the type of border that surrounds the canvas and is one of 0, *GUI.LINE*, *GUI.INDENT* or *GUI.EXDENT*. A border of 0 is the default and is the same as *GUI.LINE*. *GUI.INDENT* and *GUI.EXDENT* only display properly if the background colour has been set to *gray* using **GUI.SetBackgroundColor**. *GUI.INDENT* makes the canvas appear indented or recessed. *GUI.EXDENT* makes the canvas appear to stand out from the window.

The *mouseDown* parameter is a procedure called when the user presses the mouse button in the canvas. The *mouseDrag* parameter is a procedure called when the user drags the mouse while the mouse button is still pressed. The *mouseUp* parameter is a procedure called when the user releases the mouse button. The parameters to all three are the x and y location of the mouse where the button was pressed (dragged/released). The coordinates are given with respect to the canvas (i.e. (0, 0) is the lower-left corner of the canvas).

Example The following program draws 10 random stars in the canvas.

```
import GUI in "%oot/lib/GUI"
var canvas : int := GUI.CreateCanvas (10, 10, maxx - 20, maxy - 20)
for i : 1 .. 10
  var x : int := Rand.Int (0, maxx - 20)
  var y : int := Rand.Int (0, maxy - 20)
  var c : int := Rand.Int (0, maxcolor)
  GUI.DrawFillOval (canvas, x, y, 20, 20, c)
end for
```

Details When **GUI.CreateCanvas** or **GUI.CreateCanvasFull** is called, the newly created canvas will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

The border of the canvas is just outside the drawing surface, so **GUI.GetWidth** and **GUI.GetHeight** will return slight larger values than *width* and *height*.
When the canvas is disabled, clicking the mouse in the canvas does not call any of the *mouseDown*, *mouseDrag*, or *mouseUp* procedures. The appearance of the canvas does not change.

Details The following GUI subprograms can be called with a button as the *widgetID* parameter:

GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.Draw..., GUI.FontDraw, GUI.Pic..., GUI.SetXOR

Status Exported qualified.
This means that you can only call the function by calling **GUI.CreateCanvas**, not by calling **CreateCanvas**.

See also **GUI.Draw..., GUI.FontDraw, GUI.Pic..., and GUI.SetXOR** for drawing on a canvas.

GUI.CreateCheckBox[Full]



Syntax

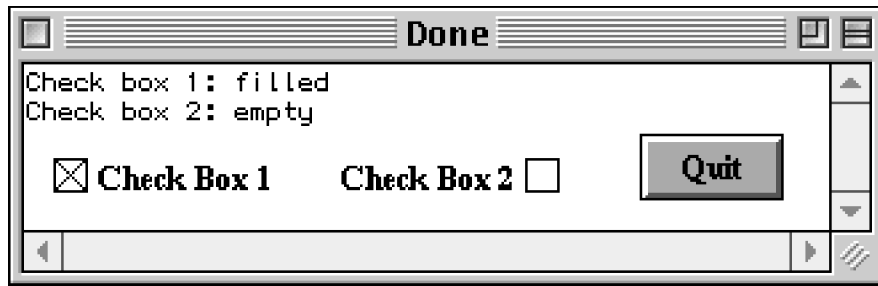
```
GUI.CreateCheckBox ( x, y : int, text : string,  
                    actionProc : procedure x (filled : boolean) ) : int
```

```
GUI.CreateCheckBoxFull ( x, y : int, text : string,  
                        actionProc : procedure x (filled : boolean),  
                        alignment : int, shortcut : char ) : int
```

Description

Creates a check box (with accompanying text) and returns the check box's widget ID.

The check box widget is used to implement a check box that can be set or unset. When you click on a check box, the status of the check box flips from set to unset and back again and the check box's *action procedure* is called with the new status as a parameter. If a check box is given a short cut, then entering the keystroke will cause the check box to change status and the *action procedure* to be called. The new status will be displayed immediately.



Two Check Boxes

The *x* and *y* parameters specify the lower-left corner of the check box (unless *alignment* is set to *GUI.RIGHT*, in which case they specify the lower-right corner of the check box). The *text* parameter specifies the text (or label) beside the check box. The *actionProc* parameter is the name of a procedure that is called when the status of the check box changes. The *actionProc* procedure must have one boolean parameter which is the new status of the check box. In **GUI.CreateCheckBox**, the check box's text is always to the right of the actual check box. In **GUI.CreateCheckBoxFull**, the text can be set to the right or left of the check box with the *alignment* parameter.

For **GUI.CreateCheckBoxFull**, the *alignment* parameter specifies the position of the check box in relation to the text as well as the meaning of the *x* and *y* parameters. The *alignment* parameter is one of 0, *GUI.LEFT*, or *GUI.RIGHT*. An *alignment* of 0 is the default and is the same as *GUI.LEFT*. *GUI.LEFT* means the actual box in the check box appears to the left of the check box's label and (*x*, *y*) specifies the lower-left corner. An *alignment* of *GUI.RIGHT* means that the actual box appears to the right of the check box's label and (*x*, *y*) specifies the lower-right corner of the check box. The *shortcut* parameter is the keystroke to be used as the button's shortcut. The *default* parameter is a boolean indicating whether the button should be the default button. If there is already a default button, and *default* is set to true, then this button becomes the new default button.

A check box's size is not specified during creation. It is determined based on the size of the text. Instead the user specifies the lower-left corner of the check box (or the lower-right if the check box is right justified).

Example The following program creates two buttons, one which draws a random circle on the screen and one which quits the program

```
import GUI in "%oot/lib/GUI"

procedure DoNothing (status : boolean)
end DoNothing

View.Set ("graphics:300;100")
var cb1 : int := GUI.CreateCheckBox (10, 10, "Check Box 1",
    DoNothing)
var cb2 : int := GUI.CreateCheckBoxFull (200, 10, "Check Box 2",
    DoNothing, GUI.RIGHT, '2')
GUI.SetCheckBox (cb2, true)
var quitBtn : int := GUI.CreateButton (230, 10, 0, "Quit", GUI.Quit)
loop
    exit when GUI.ProcessEvent
end loop
var cb1Status : boolean := GUI.GetCheckBox (cb1)
var cb2Status : boolean := GUI.GetCheckBox (cb2)
if cb1Status then
    put "Check box 1: filled"
else
    put "Check box 1: empty"
end if
if cb2Status then
    put "Check box 2: filled"
else
    put "Check box 2: empty"
end if
```

Details When **GUI.CreateButton** or **GUI.CreateButtonFull** is called, the newly created check box will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

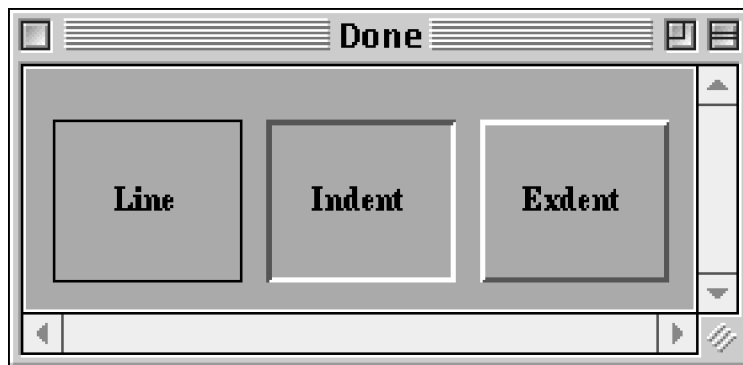
When a check box is not enabled, the label beside the check box is grayed out and the check box no longer responds to any mouse clicks or keystrokes until the check box is enabled again.

Details	<p>The following GUI subprograms can be called with a check box as the <i>widgetID</i> parameter:</p> <p>GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.SetLabel, GUI.GetCheckBox, GUI.SetCheckBox</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateCheckBox, not by calling CreateCheckBox.</p>
See also	<p>GUI.SetLabel for changing the chec box's text and GUI.GetCheckBox and GUI.SetCheckBox for reading and setting the check box's state.</p>

GUI.CreateFrame



Syntax	GUI.CreateFrame (<i>x1, y1, x2, y2, kind : int</i>) : int
Description	<p>Creates a frame and returns the frame's widget ID.</p> <p>A frame is a box drawn around other GUI widgets to make the window look better and help organize the GUI elements.</p>



Three Types of Frames With a Label in Each Frame

Frames are the only GUI widgets that can have other widgets placed within them. Frames are passive widgets, meaning that they do not respond to button clicks or keystrokes.

The *x1* and *y1* parameters specify the lower-left corner of the frame and the *x2* and *y2* parameters specify the upper-right corner of the frame. The *kind* parameter specifies the type of frame. This is one of 0, *GUI.LINE*, *GUI.INDENT*, or *GUI.EXDENT*. A *kind* of 0 is the default and is the same as *GUI.LINE*.

GUI.INDENT and *GUI.EXDENT* only display properly if the background colour has been set to *gray* using **GUI.SetBackgroundColor**. *GUI.INDENT* makes the contents frame appear indented or recessed. *GUI.EXDENT* makes the contents of the frame appear to stand out from the window.

Example The following program draws three frames in the window and draws a label in each one.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:250;90")
GUI.SetBackgroundColor (gray)
var lineFrame, indentFrame, exdentFrame : int
var lineLabel, indentLabel, exdentLabel : int
lineFrame := GUI.CreateFrame (10, 10, 80, 70, 0)
indentFrame := GUI.CreateFrame (90, 10, 160, 70, GUI.INDENT)
exdentFrame := GUI.CreateFrame (170, 10, 240, 70, GUI.EXDENT)
% Label the lines.
lineLabel := GUI.CreateLabelFull (10, 10, "Line", 70, 60,
    GUI.CENTER + GUI.MIDDLE, 0)
indentLabel := GUI.CreateLabelFull (90, 10, "Indent", 70, 60,
    GUI.CENTER + GUI.MIDDLE, 0)
exdentLabel := GUI.CreateLabelFull (170, 10, "Exdent", 70, 60,
    GUI.CENTER + GUI.MIDDLE, 0)
```

Details When **GUI.CreateFrame** is called, the newly created frame will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

A frame widget is a passive widget and cannot be enabled or disabled.

Details The following GUI subprograms can be called with a frame as the *widgetID* parameter:

**GUI.Show, GUI.Hide, GUI.Dispose,
GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight,
GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize**

Status Exported qualified.

This means that you can only call the function by calling **GUI.CreateFrame**, not by calling **CreateFrame**.

GUI.CreateHorizontalScrollBar[Full]

Syntax

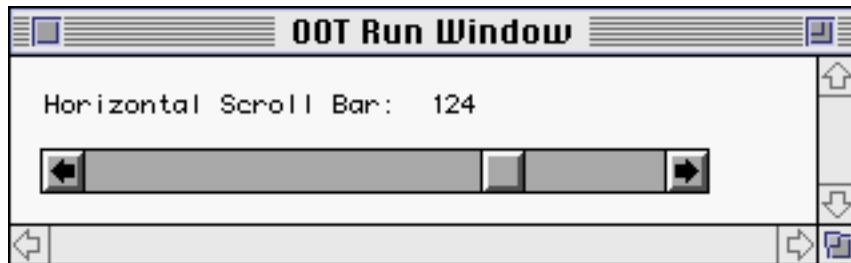
```
GUI.CreateHorizontalScrollBar ( x, y, size : int,  
                                min, max, start : int,  
                                actionProc : procedure x (value : int) ) : int
```

```
GUI.CreateHorizontalScrollBarFull ( x, y, size : int,  
                                     min, max, start : int,  
                                     actionProc : procedure x (value : int),  
                                     arrowInc, pageInc, thumbSize : int) : int
```

Description

Creates a horizontal (left-right) scroll bar and returns the scroll bar's widget ID.

A scroll bar is a widget that allows users to see a piece of a document that cannot be displayed on the screen in its entirety. The picture below shows a horizontal scroll bar. To control a scroll bar, there are a few choices: the user can click on the thumb (the box in the scroll bar) and slide it left or right, or the user can click in the scroll bar itself to the left or right of the thumb (in which case the thumb is moved up or down one "page"), or the user can click on the left or right arrows at the ends of the scroll bar (in which case the thumb is moved left or right one "arrow increment").



A Horizontal Scroll Bar

The programmer defines a page or an arrow increment. When the value of the scroll bar changes, the *action procedure* of the scroll bar is called with the new value as a parameter. The *action procedure* should then redraw the contents using the new value of the scroll bar.

The range of values that the scroll bar will give is determined by the *min* and *max* parameters in the *Create* call. The left side of the scroll bar represents the minimum value, while the right represents the maximum value. There is also the "thumb size". This represents the range of values that can be seen at once on the screen.

By default, the arrow increment (the amount the value is changed when the scrolling arrows are pressed) is set to one. The page increment (the amount the value is changed when the user clicks in the bar to the right or left of the thumb) is set to one quarter the difference between the minimum and the maximum. The "thumb size" is set to zero (see the description of scroll bars for an explanation of the thumb size).

The *x* and *y* parameters specify the lower-left corner of the scroll bar. The *size* parameter specifies the length of the scroll bar (including the arrows) in pixels. The *min* and *max* parameters are the minimum and maximum values returned by the scroll bar. The *start* parameter is the initial value of the scroll bar and should be between *min* and *max* inclusive. The *actionProc* parameter is the name of a procedure that is called when the value of the scroll bar is changed. The parameter to the *action procedure* is the current value of the scroll bar.

Example The following program creates a horizontal scroll bar. Whenever the scroll bar's value is changed, a message is displayed in the window.

```
import GUI in "%oot/lib/GUI"

View.Set ("graphics:300;60")
var scrollBar : int

procedure ScrollBarMoved (value : int)
    Text.Locate (2, 3)
    put "Horizontal Scroll Bar: ", value : 4
end ScrollBarMoved

scrollBar := GUI.CreateHorizontalScrollBar (10, 10, 250,
    50, 150, 50, ScrollBarMoved)
loop
    exit when GUI.ProcessEvent
end loop
```

Description For **GUI.CreateHorizontalScrollBarFull**, the *arrowInc* parameter specifies the arrow increment (the amount the scroll bar's value is changed when the scroll arrows are pressed). The *pageInc* specifies the page increment (the amount the scroll bar's value is changed when the user clicks in the page left/right section of the scroll bar). The *thumbSize* parameter specifies the "thumb size". (See the scroll bar explanation for more detail on a scroll bar's "thumb size").

For example, if you have a window that can display 20 lines of text at once and there are 100 lines of text, you would set *min* to 1, *max* to 100 and *thumbSize* to 20. The value returned by the scroll bar would then be the line number of the first line on the screen to be displayed. When the scroll bar was at its maximum value, it would return 81, since by doing so, lines 81-100 would be displayed.

Example Here is an example program that scrolls a large picture over a smaller window.

```
% The "ScrollPic" program.
import GUI in "%oot/lib/GUI"
```

```

var h, v : int      % The scroll bars.
var canvas : int    % The canvas.
var pic : int       % The picture.
const width : int := 220 % The width of the canvas.

procedure ScrollPic (ignore : int)
    % Get the current value of the scroll bars.
    var x : int := GUI.GetSliderValue (h)
    var y : int := GUI.GetSliderValue (v)
    GUI.PicDraw (canvas, pic, -x, -y, picCopy)
end ScrollPic

pic := Pic.FileNew ("Forest.bmp")
if pic <= 0 then
    put "Error loading picture: ", Error.LastMsg
    return
end if

View.Set ("graphics:265;265")

canvas := GUI.CreateCanvas (15, 15 + GUI.GetScrollBarWidth,
    width, width)
% Note the frame of the canvas is:
% (14, 14 + ScrollbarWidth) - (235, 235 + ScrollbarWidth)
h := GUI.CreateHorizontalScrollBarFull (14, 14,
    221, 0, Pic.GetWidth (pic), 0, ScrollPic, 3, 100, width)
v := GUI.CreateVerticalScrollBarFull (235,
    14 + GUI.GetScrollBarWidth, 221, 0, Pic.GetHeight (pic),
    Pic.GetHeight (pic), ScrollPic, 3, 100, width)
ScrollPic (0) % Draw the picture initially

loop
    exit when GUI.ProcessEvent
end loop

```

Details In some instances, you will want the the minimum and maximum values of the scroll bar to be reversed (right/top is minimum). In that case, call the **GUI.SetSliderReverse** procedure to flip the values of the scroll bar.

Scroll bars always have a fixed height (for horizontal scroll bars) or width (for vertical scroll bars). To get a scroll bar's width, use the **GUI.GetScrollBarWidth** function.

When **GUI.CreateHorizontalScrollBar** or **GUI.CreateHorizontalScrollBarFull** is called, the newly created scroll bar will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

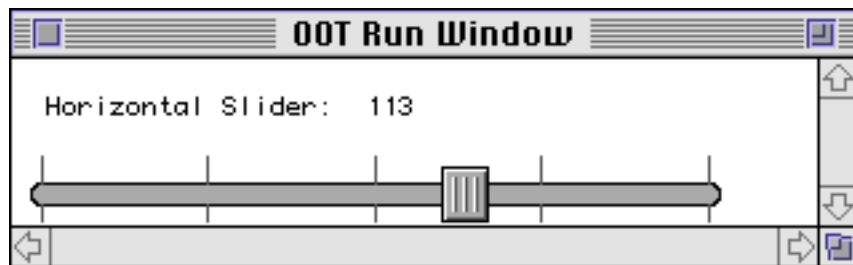
When a scroll bar is not enabled, the gray in the bar is set to white and the thumb is not displayed. The scroll bar no longer responds to any mouse clicks until the scroll bar is enabled again.

Details	<p>The following GUI subprograms can be called with a scroll bar as the <i>widgetID</i> parameter:</p> <p>GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.GetSliderValue, GUI.SetSliderValue, GUI.SetSliderMinMax, GUI.SetSliderSize, GUI.SetSliderReverse, GUI.SetScrollAmount</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateHorizontalScrollBar, not by calling CreateHorizontalScrollBar.</p>
See also	<p>GUI.GetSliderValue and GUI.SetSliderValue for reading and setting the value of a scroll bar, GUI.SetSliderMinMax for changing the minimum and maximum values of a scroll bar, and GUI.SetScrollAmount for changing the scrolling increments and thumb size of a scroll bar. See also GUI.SetSliderSize for setting the length of a scroll bar and GUI.SetSliderReverse for reversing the sense of a scroll bar.</p>

GUI.CreateHorizontalSlider



Syntax	<p>GUI.CreateHorizontalSlider (<i>x, y, length : int, min, max, start : int, actionProc : procedure x (value : int)) : int</i></p>
Description	<p>Creates a horizontal (left-right) slider and returns the slider's widget ID.</p> <p>A slider is a widget that allows the user to set a continuous set of values. It has a real-life equivalent in things such as a stereo volume control.</p>



A Horizontal Slider

To control a slider, the user clicks on the slider box and drags it back and forth. Every time the value changes, a procedure is called with the new value as a parameter.

The range of values that the slider will give is determined by the *min* and *max* parameters in the *Create* call. The left side of the slider represents the minimum value, while the right represents the maximum value.

The *x* and *y* parameters specify the lower-left corner of the slider track. This means that the slider actually extends above and below this point (and slightly to the left of it to take into account the rounded end of the track). The *length* parameter specifies the length of the track in pixels. (You can use **GUI.GetX**, **GUI.GetY**, **GUI.GetWidth**, and **GUI.GetHeight** to get the exact dimensions of the slider.) The *min* and *max* parameters are the minimum and maximum values returned by the slider. The *start* parameter is the initial value of the slider and should be between *min* and *max* inclusive. The *actionProc* parameter is the name of a procedure that is called when the value of the slider is changed. The parameter to the *action procedure* is the current value of the slider.

Example The following program creates a horizontal slider. Whenever the slider's value is changed, a message is displayed in the window.

```
import GUI in "%oot/lib/GUI"

View.Set ("graphics:300;60")
var slider : int

procedure SliderMoved (value : int)
    Text.Locate (2, 3)
    put "Horizontal Slider: ", value : 4
end SliderMoved

slider := GUI.CreateHorizontalSlider (10, 10, 250,
    50, 150, 50, SliderMoved )
loop
    exit when GUI.ProcessEvent
end loop
```

Details In some instances, you will want the the minimum and maximum values of the slider to be reversed (right is minimum). In that case, call the **GUI.SetSliderReverse** procedure to flip the values of the slider.

Sliders always have a fixed height (for horizontal sliders) or width (for vertical sliders).

When **GUI.CreateHorizontalSlider** or **GUI.CreateHorizontalSliderFull** is called, the newly created slider will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

When a slider is not enabled, the appearance does not change. However, the slider no longer responds to any mouse clicks until it is enabled again.

Details The following GUI subprograms can be called with a slider as the *widgetID* parameter:

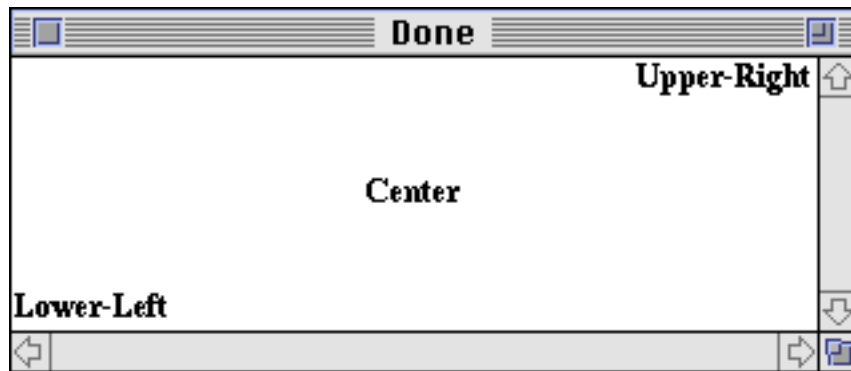
GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose,
 GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight,
 GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize,
 GUI.GetSliderValue, GUI.SetSliderValue,
 GUI.SetSliderMinMax, GUI.SetSliderSize,
 GUI.SetSliderReverse

Status	Exported qualified. This means that you can only call the function by calling GUI.CreateHorizontalSlider , not by calling CreateHorizontalSlider .
See also	GUI.GetSliderValue and GUI.SetSliderValue for reading and setting the value of a slider, GUI.SetSliderMinMax for changing the minimum and maximum values of a slider. See also GUI.SetSliderSize for setting the length of a slider and GUI.SetSliderReverse for reversing the sense of a slider.

GUI.CreateLabel[Full]



Syntax	GUI.CreateLabel (<i>x</i> , <i>y</i> : int , <i>text</i> : string) : int GUI.CreateLabelFull (<i>x</i> , <i>y</i> : int , <i>text</i> : string , <i>width</i> , <i>height</i> , <i>alignment</i> , <i>fontID</i> : int) : int
Description	Creates a label and returns the label's widget ID. The label widget is used to display text. It can be used to display text in a variety of fonts and sizes. Label widgets can also be aligned in a variety of ways.



Three Labels

The x and y parameters specify the lower-left corner of the area in which the text will be drawn. For **GUI.CreateLabel**, this is the lower-left corner of the text. The *text* parameter specifies the text of the label.

For **GUI.CreateLabelFull**, the *width* and *height* parameters specify the area in which the label is to appear. This is used for alignment purposes. See the program below for an example of aligning the text to different corners of the window. The *alignment* parameter specifies the alignment of the text in the text area. This value is the sum of horizontal alignment and the vertical alignment. The horizontal alignment is one of 0, *GUI.LEFT*, *GUI.CENTER*, or *GUI.RIGHT*. A horizontal alignment of 0 is the default and is the same as the alignment of *GUI.LEFT*. The vertical alignment is one of 0, *GUI.TOP*, *GUI.MIDDLE*, or *GUI.BOTTOM*. A horizontal alignment of 0 is the default and is the same as the alignment of *GUI.BOTTOM*. These alignments align the text in various ways in the text area. The *fontID* parameter specifies the font ID of the font to be used in the text field. The font ID is received from a *Font.New* call. Do not call *Font.Free* for this font ID until the label has been disposed of by calling **GUI.Dispose**.

By using the *fontID* parameter, labels can be have any size or typeface.

Labels are passive widgets, meaning that they do not respond to button clicks or keystrokes.

Example The following program creates three labels, one with the default alignment, the other two aligned to appear in the center and upper-right corner of the window.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:300;100")
var lowerLeft : int := GUI.CreateLabel (0, 0, "Lower-Left")
var center : int := GUI.CreateLabelFull (0, 0, "Center", maxx, maxy,
    GUI.MIDDLE + GUI.CENTER, 0)
var upperRight : int := GUI.CreateLabelFull (0, 0, "Upper-Right",
    maxx, maxy, GUI.RIGHT + GUI.TOP, 0)
```

Details When **GUI.CreateLabel** or **GUI.CreateLabelFull** is called, the newly created label will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

A frame widget is a passive widget and cannot be enabled or disabled.

Details The following GUI subprograms can be called with a label as the *widgetID* parameter:

```
GUI.Show, GUI.Hide, GUI.Dispose,
GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight,
GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize,
GUI.GetSliderValue, GUI.SetSliderValue,
GUI.SetSliderMinMax, GUI.SetLabel
```


Example The following program draws three frames in the window.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:250;90")
GUI.SetBackgroundColor (gray)
var lineFrame, indentFrame, exdentFrame : int
lineFrame := GUI.CreateFrame (10, 10, 80, 70, 0, "Line")
indentFrame := GUI.CreateFrame (90, 10, 160, 70, GUI.INDENT,
    "Indent")
exdentFrame := GUI.CreateFrame (170, 10, 240, 70, GUI.EXDENT,
    "Exdent")
```

Details When **GUI.CreateLabelledFrame** is called, the newly created labelled frame will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

A labelled frame widget is a passive widget and cannot be enabled or disabled.

Details The following GUI subprograms can be called with a labelled frame as the *widgetID* parameter:

```
GUI.Show, GUI.Hide, GUI.Dispose,
GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight,
GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize,
GUI.SetLabel
```

Status Exported qualified.

This means that you can only call the function by calling **GUI.CreateLabelledFrame**, not by calling **CreateLabelledFrame**.

See also **GUI.SetLabel** for changing the frame's text.

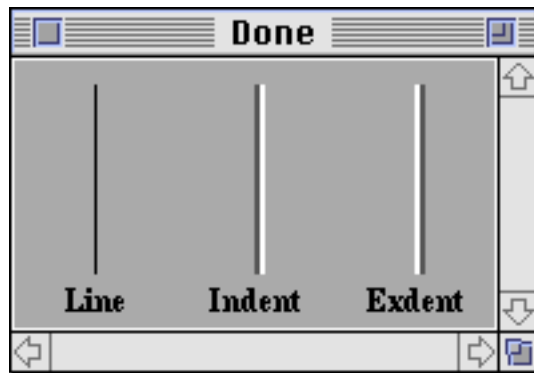
GUI.CreateLine



Syntax **GUI.CreateLine** (*x1, y1, x2, y2, kind* : int) : int

Description Creates a line and returns the line's widget ID.

Lines are generally used to separate parts of a window. A line is used to make the window look better and help organize the GUI elements.



Three Types of Lines

Lines are passive widgets, meaning that they do not respond to button clicks or keystrokes.

The *x1* and *y1* parameters specify one end-point of the line and the *x2* and *y2* parameters specify the other end point. The line must either be horizontal or vertical (i.e. *x1* must equal *x2* or *y1* must equal *y2*). The *kind* parameter specifies the type of line. This is one of 0, *GUI.LINE*, *GUI.INDENT* or *GUI.EXDENT*. A *kind* of 0 is the default and is the same as *GUI.LINE*.

GUI.INDENT and *GUI.EXDENT* only display properly if the background colour has been set to *gray* using **GUI.SetBackgroundColor**. *GUI.INDENT* makes the line appear indented or recessed. *GUI.EXDENT* makes the line appear to stand out from the window.

Example The following program draws three lines with three labels in the window.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:180;100")
GUI.SetBackgroundColor (gray)
var line indentLine, exdentLine : int
var lineLabel, indentLabel, exdentLabel : int

line := GUI.CreateLine (30, 20, 30, 90, 0)
indentLine := GUI.CreateLine (90, 20, 90, 90, GUI.INDENT)
exdentLine := GUI.CreateLine (150, 20, 150, 90, GUI.EXDENT)
% Label the lines.
lineLabel := GUI.CreateLabelFull (30, 15, "Line", 0, 0,
    GUI.CENTER + GUI.TOP, 0)
indentLabel := GUI.CreateLabelFull (90, 15, "Indent", 0, 0,
    GUI.CENTER + GUI.TOP, 0)
exdentLabel := GUI.CreateLabelFull (150, 15, "Exdent", 0, 0,
    GUI.CENTER + GUI.TOP, 0)
```

Details When **GUI.CreateLine** is called, the newly created line will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

A line widget is a passive widget and cannot be enabled or disabled.

Details The following GUI subprograms can be called with a line as the *widgetID* parameter:

**GUI.Show, GUI.Hide, GUI.Dispose,
GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight,
GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize**

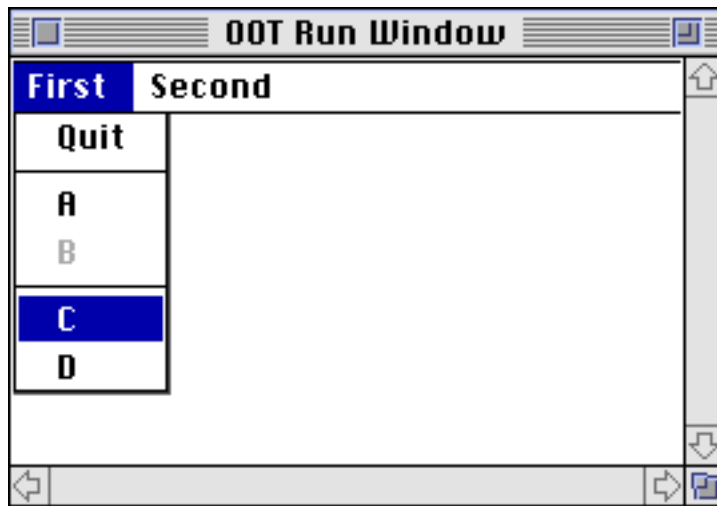
Status Exported qualified.
This means that you can only call the function by calling **GUI.CreateLine**, not by calling **CreateLine**.

GUI.CreateMenu

Win DOS Mac
● ● ●
○ × ×
X TOOT Term

Syntax **GUI.CreateMenu (*name* : string) : int**

Description Creates a menu and returns the menu's widget ID. The menu will be added after the other menus in the menu bar. If there are no previous menus, then a menu bar is automatically created and the menu added.
The *name* parameter specifies the text that appears in the menu bar.



A Menu With an Item Selected

Menus are used in most modern interfaces. In order to create a full set of menus, you must create the menu and then create the menu items in that menu. The menus are automatically added to the menu bar of the selected menu.

As of the v1.0 release of the GUI Library, it is an error to create a menu item without having created a menu first. In future releases it will be possible to create menus and attach and remove them from menu bars when desired.

Example The following program creates a series of menus with menu items in them. It then disables the second menu.

```

import GUI in "%oot/lib/GUI"

View.Set ("graphics:250;150")
var first, second : int            % The menus.
var item : array 1 .. 12 of int    % The menu items.

var name : array 1 .. 12 of string (20) :=
  init ("Quit", "---", "A", "B", "---", "C", "D",
    "Disable B Menu Item", "Enable B Menu Item", "---",
    "Disable Second Menu", "Enable Second Menu")

procedure MenuSelected
  for i : 1 .. 12
    if item (i) = GUI.GetEventWidgetID then
      Text.Locate (maxrow, 1)
      put name (i) + " selected"        " ..
    end if
  end for
end MenuSelected

procedure DisableB
  GUI.Disable (item (4))
end DisableB

procedure EnableB
  GUI.Enable (item (4))
end EnableB

procedure DisableFirst
  GUI.Disable (first)
end DisableFirst

procedure EnableFirst
  GUI.Enable (first)
end EnableFirst

% Create the menus
first := GUI.CreateMenu ("First")
item (1) := GUI.CreateMenuItem (name (1), GUI.Quit)
for cnt : 2 .. 7
  item (cnt) := GUI.CreateMenuItem (name (cnt),
    MenuSelected)
end for

second := GUI.CreateMenu ("Second")
item (8) := GUI.CreateMenuItem (name (8), DisableB)
item (9) := GUI.CreateMenuItem (name (9), EnableB)
item (10) := GUI.CreateMenuItem (name (10), MenuSelected)

```

	<pre> item (11) := GUI.CreateMenuItem (name (11), DisableFirst) item (12) := GUI.CreateMenuItem (name (12), EnableFirst) loop exit when GUI.ProcessEvent end loop </pre>
Details	When a menu is not enabled, the text in the menu bar is grayed out and clicking on the menu does not cause the menu to appear.
Details	<p>The following GUI subprograms can be called with a menu as the <i>widgetID</i> parameter:</p> <p>GUI.Show, GUI.Hide, GUI.Dispose, GUI.Enable, GUI.Disable</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateMenu, not by calling CreateMenu.</p>
See also	GUI.CreateMenuItem for adding items to a menu. See also GUI.ShowMenuBar and GUI.HideMenuBar for showing and hiding the menu bar.

GUI.CreateMenuItem[Full]



Syntax	<pre> GUI.CreateMenuItem (<i>name</i> : string, <i>actionProc</i> : procedure <i>x</i> ()) : int GUI.CreateMenuItemFull (<i>name</i> : string, <i>actionProc</i> : procedure <i>x</i> (), <i>shortCut</i> : char, <i>addNow</i> : boolean) : int </pre>
Description	<p>Creates a menu item and returns the menu item's widget ID.</p> <p>Menu items are the individual entries of a menu. To create menus for a window, you must create a menu, then create the menu items for that menu, then create the next menu, etc. All menu items are automatically added to the last menu and after the last menu item of the currently selected (not active!) window.</p> <p>The menu item will be added to the last menu after the other menu items in the menu. If there are no menus defined, an error results.</p>

The *name* parameter specifies the text that is to appear. A *name* of three dashes ("---") creates a separator across the menu. The *actionProc* parameter specifies the name of a procedure to be called when user selects the menu item from the menu.

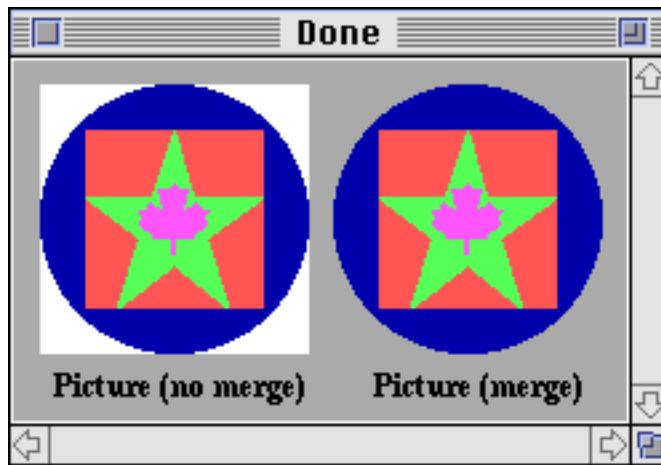
For **GUI.CreateMenuItemFull**, the *shortCut* parameter specifies the keystroke to be used as the menu item's shortcut. If no shortcut is desired, then '\0' can be used. The *addNow* parameter has no effect in the current version of the GUI Library. In future versions, it will allow you to create menu items that can then be added to a menu later in the program.

Examples	See the example for GUI.CreateMenu .
Details	When a menu item is not enabled, the text of the menu item is grayed out and clicking on the menu item does not cause the menu to appear.
Details	The following GUI subprograms can be called with a menu as the <i>widgetID</i> parameter: GUI.Show , GUI.Hide , GUI.Dispose , GUI.Enable , GUI.Disable
Status	Exported qualified. This means that you can only call the function by calling GUI.CreateMenuItem , not by calling CreateMenuItem .

GUI.CreatePicture



Syntax	GUI.CreatePicture (<i>x</i> , <i>y</i> , <i>picture</i> : int , <i>mergePic</i> : boolean) : int
Description	Creates a picture and returns the picture's widget ID. The picture widget is used to display a picture. It can be used to display a picture either merged into the background or not. The <i>x</i> and <i>y</i> parameters specify the lower-left corner of the picture. The <i>picture</i> parameter specifies the picture ID of the picture. The picture ID is received from a Pic.New or Pic.FileNew call. Do not call Pic.Free for this picture ID until the button has been disposed of by calling GUI.Dispose . The <i>mergePic</i> parameter is a boolean that specifies whether anything that was the background colour in the picture (usually colour 0) should be set to the background colour of the window. A picture widget is a passive widget and cannot be enabled or disabled.



Two Pictures

Example The following program draws two pictures, merged and not merged.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:230;135")

% We'll need to create a picture for our Picture widget. Normally
% an external file (and Pic.FileNew) would be used.
Draw.FillOval (50, 50, 50, 50, blue)
Draw.FillBox (17, 17, 83, 83, brightred)
Draw.FillStar (17, 17, 83, 83, brightgreen)
Draw.FillMapleLeaf (37, 37, 63, 63, brightpurple)
var pic : int := Pic.New (0, 0, 100, 100)

var picture1, picture2 : int
var label1, label2 : int

GUI.SetBackgroundColor (gray)

label1 := GUI.CreateLabel (15, 5, "Picture (no merge)")
picture1 := GUI.CreatePicture (10, 25, pic, false)

label2 := GUI.CreateLabel (135, 5, "Picture (merge)")
picture2 := GUI.CreatePicture (120, 25, pic, true)
```

Details When **GUI.CreatePicture** is called, the newly created picture will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

A picture widget is a passive widget and cannot be enabled or disabled.

Details The following GUI subprograms can be called with a picture as the *widgetID* parameter:

GUI.Show, GUI.Hide, GUI.Dispose,
GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight,
GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize

Status	Exported qualified. This means that you can only call the function by calling GUI.CreatePicture , not by calling CreatePicture .
--------	---



GUI.CreatePictureButton[Full]

Syntax	GUI.CreatePictureButton (<i>x</i> , <i>y</i> , <i>picture</i> : int , <i>actionProc</i> : procedure <i>x</i> ()) : int
	GUI.CreatePictureButtonFull (<i>x</i> , <i>y</i> , <i>picture</i> : int , <i>actionProc</i> : procedure <i>x</i> (), <i>width</i> , <i>height</i> : int , <i>shortcut</i> : char , <i>mergePic</i> : boolean) : int
Description	<p>Creates a picture button and returns the button's widget ID.</p> <p>Picture buttons behave like buttons (see GUI.CreateButton) except that instead of text on the button, a picture specified by the user is displayed on the button. The picture button widget responds to mouse clicks and keystrokes in the same manner as a regular button widget.</p> <p>The picture must be created by the program beforehand using Pic.New or Pic.FileNew. The resulting picture can then be used as a parameter to GUI.CreatePictureButton. In general, pictures should be a maximum of about 50 pixels high and wide, although there is no built-in limit in the GUI library.</p> <p>The <i>x</i> and <i>y</i> parameters specify the lower-left corner of the picture button. The <i>picture</i> parameter specifies the picture ID of the picture to be displayed on the button. (Note that, in general, this picture should be fairly small.) The picture ID is received from a Pic.New or Pic.FileNew call. Do not call Pic.Free for this picture ID until the button has been disposed of by calling GUI.Dispose. The <i>actionProc</i> parameter specifies the name of a procedure that is called when the picture button is pressed.</p> <p>For GUI.CreatePictureButtonFull, the <i>width</i> and <i>height</i> parameters specify the width and height of the button. If they are set to 0, then the picture radio button is automatically sized to fit the picture. If you need to know the precise size of the button, use the GUI.GetWidth and GUI.GetHeight functions. If <i>width</i> and <i>height</i> are larger than the picture, the picture is centered in the button. The <i>shortCut</i> parameter is the keystroke to be used as the button's shortcut. The <i>mergePic</i> parameter specifies whether anything that was the background colour in the picture (usually colour 0) should be set to the background colour of the button (which is usually gray). This defaults to true for <i>CreatePictureButton</i>.</p>



Two Picture Buttons

Example The following program displays five picture buttons which output a message when pressed.

```

import GUI in "%oot/lib/GUI"
View.Set ("graphics:100;70")

const size : int := 25     % The buttons size.
const border : int := 3

var starButton, mapleButton, starPic, mapleLeafPic : int

procedure StarPressed
  Text.Locate (1, 1)
  put "Star Pressed "
end StarPressed

procedure MaplePressed
  Text.Locate (1, 1)
  put "Maple Pressed "
end MaplePressed

% Create the pictures.
% The star.
Draw.Star (border, border, border + size, border + size, black)
Draw.Star (border + 1, border + 1, border + size - 1,
  border + size - 1, black)
Draw.FillStar (border + 2, border + 2, border + size - 2,
  border + size - 2, brightred)
starPic := Pic.New (0, 0, 2 * border + size, 2 * border + size)

% The mapleleaf.
Draw.FillBox (border, border, border + size, border + size, white)
Draw.MapleLeaf (border, border, border + size, border + size, black)
Draw.MapleLeaf (border + 1, border + 1, border + size - 1,
  border + size - 1, black)
Draw.FillMapleLeaf (border + 2, border + 2, border + size - 2,
  border + size - 2, brightred)
mapleLeafPic := Pic.New (0, 0, 2 * border + size, 2 * border + size)

% Create the picture buttons.
Draw.Cls
starButton := GUI.CreatePictureButton (10, 10, starPic, StarPressed)
  
```

```
mapleButton := GUI.CreatePictureButton (55, 10, mapleLeafPic,
MaplePressed)
```

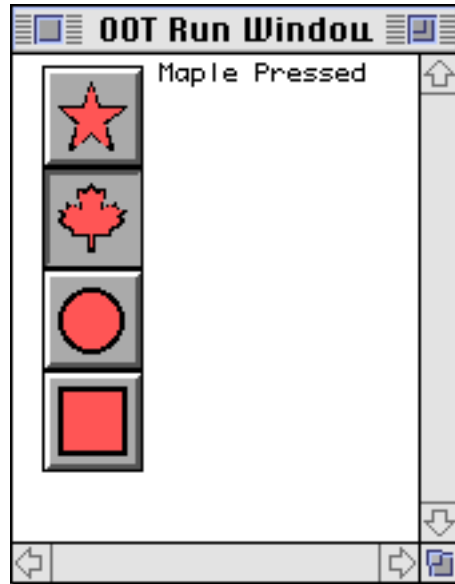
```
loop
  exit when GUI.ProcessEvent
end loop
```

Details	<p>When GUI.CreatePictureButton or GUI.CreatePictureButtonFull is called, the newly created picture will be displayed immediately unless GUI.DisplayWhenCreated has been called with the <i>display</i> parameter set to false.</p> <p>When a picture button is not enabled, the picture button is grayed out and the picture button no longer responds to any mouse clicks or keystrokes until the button is enabled again.</p>
Details	<p>The following GUI subprograms can be called with a picture button as the <i>widgetID</i> parameter:</p> <p>GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreatePictureButton, not by calling CreatePictureButton.</p>

GUI.CreatePictureRadioButton[Full]

Syntax	<pre>GUI.CreatePictureRadioButton (<i>x, y, picture</i> : int, <i>joinID</i> : int, <i>actionProc</i> : procedure <i>x</i> ()) : int</pre> <pre>GUI.CreatePictureRadioButtonFull (<i>x, y, picture</i> : int, <i>joinID</i> : int, <i>actionProc</i> : procedure <i>x</i> () , <i>width, height</i> : int, <i>shortcut</i> : char, <i>mergePic</i> : boolean) : int</pre>
Description	<p>Creates a picture radio button and returns the button's widget ID.</p> <p>Picture radio buttons behave like picture buttons (see GUI.CreatePictureButton) except that they have the "radio" property. That is, one of the buttons in the radio group is always selected, and if another button in the group is selected, the previously selected button is unselected.</p>

A common example is the buttons on a paint program that indicate the current shape being painted. The maple leaf button is currently selected. If the star button is selected by the user, the maple leaf button becomes unselected. For picture buttons, the selected button appears depressed.



Four Picture Radio Buttons with the Maple Leaf Selected

A radio group is created by first creating a single radio button. To add another button to the group, a second radio button is created specifying the first radio button in the *joinID* parameter. Subsequent radio buttons are added, each specifying a previous member of the group in the *joinID* parameter.

The picture must be created by the program beforehand using **Pic.New** or **Pic.FileNew**. The resulting picture can then be used as a parameter to **GUI.CreatePictureButton**. In general, pictures should be a maximum of about 50 pixels high and wide, although there is no built-in limit in the GUI library.

The *x* and *y* parameters specify the lower-left corner of the picture radio button. If these are both -1 and *joinID* is not zero, then the button will be placed directly below the previous picture radio button in the group. The *picture* parameter specifies the picture ID of the picture to be displayed on the button. (Note that, in general, this picture should be fairly small.) The picture ID is received from a **Pic.New** or **Pic.FileNew** call. Do not call **Pic.Free** for this picture ID until the button has been disposed of by calling **GUI.Dispose**. The *joinID* parameter specifies a member of the radio group that this widget should join. A *joinID* of 0 specifies this radio button is not a member of any group. The *actionProc* parameter specifies the name of a procedure that is called when the picture button is pressed.

For **GUI.CreatePictureRadioButtonFull**, the *width* and *height* parameters specify the width and height of the button. If they are set to 0, then the picture radio button is automatically sized to fit the picture. If you need to know the precise size of the button, use the **GUI.GetWidth** and **GUI.GetHeight** functions. If *width* and *height* are larger than the picture, the picture is centered in the button. The *shortCut* parameter is the keystroke to be used as the button's shortcut. The *mergePic* parameter specifies whether anything that was the background colour in the picture (usually colour 0) should be set to the background colour of the button (which is usually gray). This defaults to true for *CreatePictureRadioButton*.

Example The following program creates and displays for picture radio buttons.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:100;70")

const size : int := 25      % The buttons size.
const border : int := 3

var starButton, mapleButton, circleButton, squareButton : int
var starPic, mapleLeafPic, circlePic, squarePic : int

procedure StarPressed
  Text.Locate (1, 1)
  put "Star Pressed "
end StarPressed

procedure MaplePressed
  Text.Locate (1, 1)
  put "Maple Pressed "
end MaplePressed

procedure CirclePressed
  Text.Locate (1, 10)
  put "Circle Pressed"
end CirclePressed

procedure SquarePressed
  Text.Locate (1, 10)
  put "Square Pressed"
end SquarePressed

% Create the pictures.
% The star.
Draw.Star (border, border, border + size, border + size, black)
Draw.Star (border + 1, border + 1, border + size - 1,
  border + size - 1, black)
Draw.FillStar (border + 2, border + 2, border + size - 2,
  border + size - 2, brightred)
starPic := Pic.New (0, 0, 2 * border + size, 2 * border + size)

% The mapleleaf.
Draw.FillBox (border, border, border + size, border + size, white)
Draw.MapleLeaf (border, border, border + size, border + size, black)
Draw.MapleLeaf (border + 1, border + 1, border + size - 1,
```

```

    border + size - 1, black)
Draw.FillMapleLeaf (border + 2, border + 2, border + size - 2,
    border + size - 2, brightred)
mapleLeafPic := Pic.New (0, 0, 2 * border + size, 2 * border + size)

% The circle.
const radius : int := size div 2
Draw.FillBox (border, border, border + size, border + size, white)
Draw.Oval (border + radius, border + radius, radius, radius, black)
Draw.Oval (border + radius, border + radius, radius - 1, radius - 1,
    black)
Draw.FillOval (border + radius, border + radius, radius - 2,
    radius - 2, brightred)
circlePic := Pic.New (0, 0, 2 * border + size, 2 * border + size)

% The square.
Draw.FillBox (border, border, border + size, border + size, white)
Draw.Box (border, border, border + size, border + size, black)
Draw.Box (border + 1, border + 1, border + size - 1,
    border + size - 1, black)
Draw.FillBox (border + 2, border + 2, border + size - 2,
    border + size - 2, brightred)
squarePic := Pic.New (0, 0, 2 * border + size, 2 * border + size)

% Create the picture buttons.
Draw.Cls
starButton := GUI.CreatePictureButton (10, maxy - 40, starPic,
    0, StarPressed)
mapleButton := GUI.CreatePictureButton (-1, -1, mapleLeafPic,
    starButton, MaplePressed)
circleButton := GUI.CreatePictureRadioButton (-1, -1, circlePic,
    mapleButton, CirclePressed)
squareButton := GUI.CreatePictureRadioButton (-1, -1, squarePic,
    circleButton, SquarePressed)

loop
    exit when GUI.ProcessEvent
end loop

```

Details When **GUI.CreatePictureRadioButton** or **GUI.CreatePictureRadioButtonFull** is called, the newly created picture will be displayed immediately unless **GUI.DisplayWhenCreated** has been called with the *display* parameter set to false.

When a picture radio button is not enabled, the picture radio button is grayed out and the picture button no longer responds to any mouse clicks or keystrokes until the button is enabled again.

Details The following GUI subprograms can be called with a picture radio button as the *widgetID* parameter:

GUI.Show, **GUI.Hide**, **GUI.Enable**, **GUI.Disable**, **GUI.Dispose**,
GUI.GetX, **GUI.GetY**, **GUI.GetWidth**, **GUI.GetHeight**,
GUI.SetPosition, **GUI.SetSize**, **GUI.SetPositionAndSize**,
GUI.SelectRadio

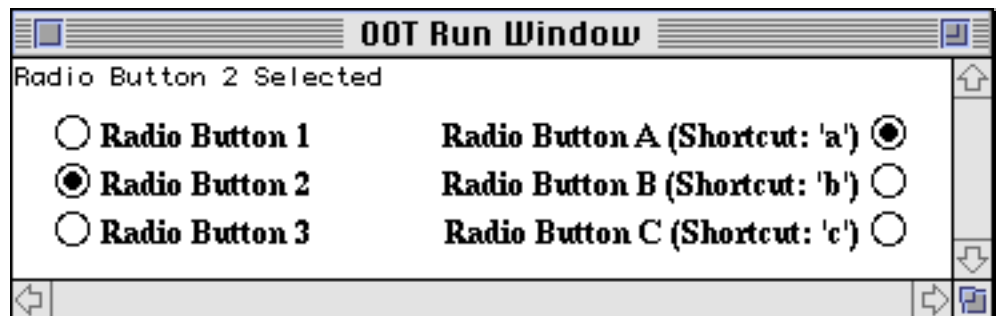
Status	Exported qualified. This means that you can only call the function by calling GUI.CreatePictureRadioButton , not by calling CreatePictureRadioButton .
See also	GUI.SelectRadio for selecting a picture radio button in a program. See also GUI.CreatePictureButton and GUI.CreateRadioButton for information on picture buttons and radio buttons.

GUI.CreateRadioButton[Full]



Syntax	GUI.CreateRadioButton (<i>x, y : int, text : string,</i> <i>joinID : int, actionProc : procedure x ()</i>) : int GUI.CreateRadioButtonFull (<i>x, y : int, text : string,</i> <i>joinID : int, actionProc : procedure x (),</i> <i>alignment : int, shortCut : char</i>) : int
--------	--

Description	<p>Creates a radio button and returns the radio button's widget ID.</p> <p>A slider is a widget that allows the user to select one of a set of values. It has a real-life equivalent in the old car stereos where a single station is selected at a time. That is, one of the buttons in the radio group is always selected, and if another button in the group is selected, the previously selected button is unselected.</p>
-------------	--



Six Radio Buttons in Two Groups

A radio group is created by first creating a single radio button. To add another button to the group, a second radio button is created specifying the first radio button in the *joinID* parameter. Subsequent radio buttons are added, each specifying a previous member of the group in the *joinID* parameter.

The *x* and *y* parameters specify the lower-left corner of the radio button (unless *alignment* is set to *GUI.RIGHT*, in which case they specify the lower-right corner of the radio button). If these are both -1 and *joinID* is not zero, then the button will be placed directly below the previous radio button in the group. The *text* parameter specifies the text (or label) beside the radio button. The *joinID* parameter specifies a member of the radio group that this widget should join. A *joinID* of 0 specifies this radio button is not a member of any group. The *actionProc* parameter is the name of a procedure that is called when the radio button is selected. In **GUI.CreateRadioButton**, the radio button's text is always to the right of the actual radio button. In **GUI.CreateRadioButtonFull**, the text can be set to the right or left of the radio button with the *alignment* parameter.

For **GUI.CreateRadioButtonFull**, the *alignment* parameter specifies the position of the radio button in relation to the text as well as the meaning of the *x* and *y* parameters. The *alignment* parameter is one of 0, *GUI.LEFT*, or *GUI.RIGHT*. An *alignment* of 0 is the default and is the same as *GUI.LEFT*. *GUI.LEFT* means the actual box in the check box appears to the left of the check box's label and (*x*, *y*) specify the lower-left corner. An *alignment* of *GUI.RIGHT* means that the actual box appears to the right of the radio button's label and (*x*, *y*) specify the lower-right corner of the radio button. The *shortcut* parameter is the keystroke to be used as the button's shortcut.

A radio button's size is not specified during creation. It is determined based on the size of the text. Instead the user specifies the lower-left corner of the radio button (or the lower-right if the radio button is right justified).

Example The following program creates six radio buttons in two groups.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:350;80")

var radio : array 1 .. 6 of int % The radio button IDs.

procedure RadioPressed
  Text.Locate (1, 1)
  put "Radio Button " ..
  for i : 1 .. 6
    if radio (i) = GUI.GetEventWidgetID then
      put i ..
    end if
  end for
  put " Selected"
end RadioPressed

radio (1) := GUI.CreateRadioButton (15, maxy - 35,
  "Radio Button 1", 0, RadioPressed)
radio (2) := GUI.CreateRadioButton (-1, -1, "Radio Button 2",
  radio (1), RadioPressed)
```



```

radio (3) := GUI.CreateRadioButton ( -1, -1, "Radio Button 3",
radio (2), RadioPressed)
radio (4) := GUI.CreateRadioButtonFull (maxx - 15, maxy - 35,
"Radio Button A (Shortcut: 'a')", 0, RadioPressed,
GUI.RIGHT, 'a')
radio (5) := GUI.CreateRadioButtonFull ( -1, -1,
"Radio Button B (Shortcut: 'b')", radio (4), RadioPressed,
GUI.RIGHT, 'b')
radio (6) := GUI.CreateRadioButtonFull ( -1, -1,
"Radio Button C (Shortcut: 'c')", radio (5), RadioPressed,
GUI.RIGHT, 'c')

```

```

loop
  exit when GUI.ProcessEvent
end loop

```

Details	<p>When a group of radio buttons is selected, the first radio button created in the group will be the selected one. You can change this by using the GUI.SelectRadio procedure to select a different one.</p> <p>When GUI.CreateRadioButton or GUI.CreateRadioButtonFull is called, the newly created picture will be displayed immediately unless GUI.DisplayWhenCreated has been called with the <i>display</i> parameter set to false.</p> <p>When a radio button is not enabled, the radio button is grayed out and the radio button no longer responds to any mouse clicks or keystrokes until the button is enabled again.</p>
Details	<p>The following GUI subprograms can be called with a radio button as the <i>widgetID</i> parameter:</p> <p>GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.SetLabel, GUI.SelectRadio</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateRadioButton, not by calling CreateRadioButton.</p>
See also	<p>GUI.SelectRadio for selecting a radio button in a program. See also GUI.SetLabel for changing the radio button's text.</p>



GUI.CreateTextBox[Full]

Syntax

GUI.CreateTextBox (*x*, *y*, *width*, *height* : **int**) : **int**

GUI.CreateTextBoxFull (*x*, *y*, *width*, *height* : **int**,
border, *fontID* : **int**) : **int**

Description

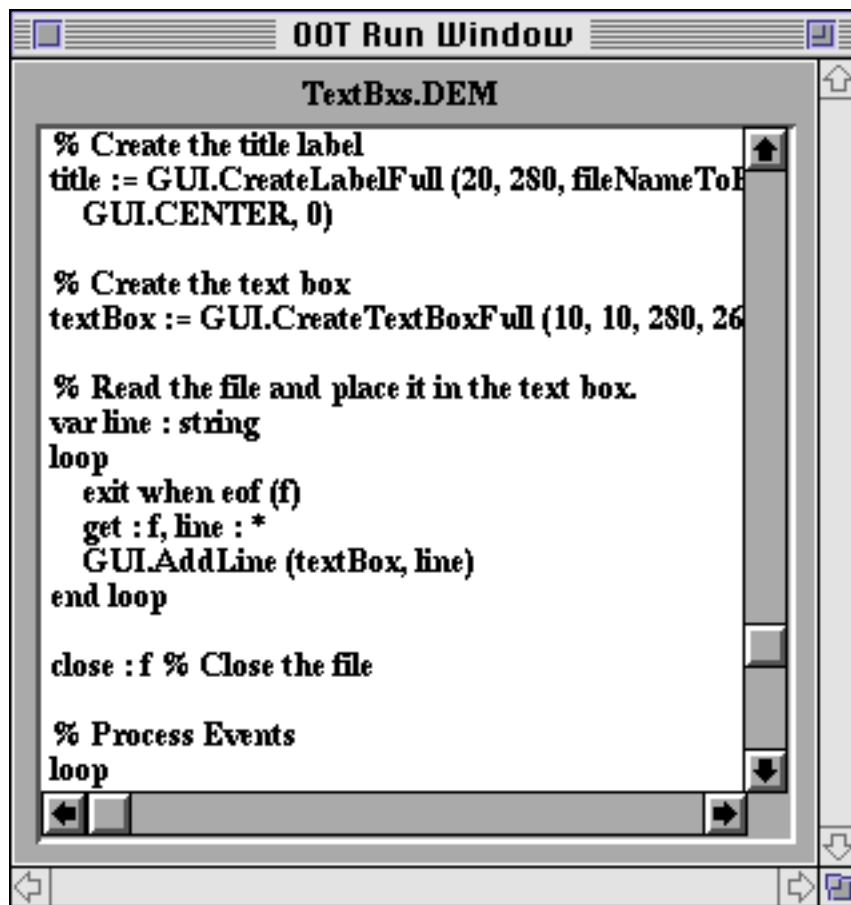
Creates a text box and returns the text box's widget ID.

A text box is a box used for displaying text. It has scroll bars that activate when text appears outside the border of the text box. The user cannot directly select, edit or modify the text in the text box.

The *x* and *y* parameters specify the lower-left corner of the area in which the text will be drawn. The *width* and *height* parameters specify the width and height of the text drawing area. The text box border is just outside the text drawing area. Because of this, **GUI.GetX** and **GUI.GetY** will return a value slightly smaller than *x* and *y* and **GUI.GetWidth** and **GUI.GetHeight** will return values slightly larger than *width* and *height*.

For **GUI.CreateTextBox**, the border around the text box is always a line. For **GUI.CreateTextBoxFull**, the type of border is specified by the *border* parameter. The *border* parameter is one of 0, *GUI.LINE*, *GUI.INDENT*, or *GUI.EXDENT*. A border of 0 is the default and is the same as *GUI.LINE*. *GUI.INDENT* and *GUI.EXDENT* only display properly if the background colour has been set to *gray* using **GUI.SetBackgroundColor**. *GUI.INDENT* makes the text box appear indented or recessed. *GUI.EXDENT* makes the text box appear to stand out from the window. The *fontID* parameter specifies the font ID of the font to be used in the text box. The font ID is received from a *Font.New* call. Do not call *Font.Free* for this font ID until the label has been disposed of by calling **GUI.Dispose**.

By using the *fontID* parameter, text boxes can have any size or typeface.



A text box displaying the contents of a file.

Example The following program displays the contents of a file in a text box.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:300;300")

const fileNameToBeViewed : string := "TextBxs.DEM"
var textBox : int           % The Text Field ID.
var title : int            % The label for the title.
var f : int                % The stream number of the file.
var line : string          % Lines to be read from the file.

% Open the file.
open : f, fileNameToBeViewed, get
if f = 0 then
    put "Unable to open " + fileNameToBeViewed + " : ",
        Error.LastMsg
    return
end if
```

```

% Set background color to gray for indented text box.
GUI.SetBackgroundColor (gray)

% Create the title label and text box.
title := GUI.CreateLabelFull (20, 280, fileNameToBeViewed, 250, 0,
    GUI.CENTER, 0)
textBox := GUI.CreateTextBoxFull (10, 10, 280, 265,
    GUI.INDENT, 0)

% Read the file and place it in the text box.
loop
    exit when eof (f)
    get : f, line : *
    GUI.AddLine (textBox, line)
end loop

close : f % Close the file.

loop
    exit when GUI.ProcessEvent
end loop

```

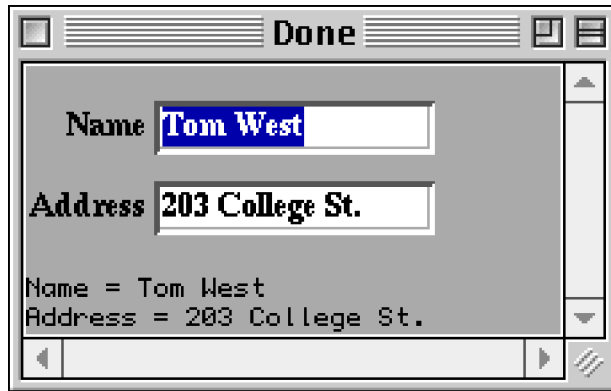
Details	<p>When GUI.CreateTextBox or GUI.CreateTextBoxFull is called, the newly created picture will be displayed immediately unless GUI.DisplayWhenCreated has been called with the <i>display</i> parameter set to false.</p> <p>A text box widget is a passive widget and cannot be enabled or disabled.</p>
Details	<p>The following GUI subprograms can be called with a text box as the <i>widgetID</i> parameter:</p> <p>GUI.Show, GUI.Hide, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.AddLine, GUI.AddText, GUI.ClearText</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateTextBox, not by calling CreateTextBox.</p>
See also	<p>GUI.AddLine, GUI.AddText for adding text to the text box. See also GUI.ClearText for clearing the text box.</p>

GUI.CreateTextField[Full]



Syntax	<pre>GUI.CreateTextField (<i>x</i>, <i>y</i>, <i>width</i> : int, <i>text</i> : string, <i>actionProc</i> : procedure <i>x</i> (<i>text</i> : string)) : int</pre> <pre>GUI.CreateTextFieldFull (<i>x</i>, <i>y</i>, <i>width</i> : int, <i>text</i> : string, <i>actionProc</i> : procedure <i>x</i> (<i>text</i> : string), <i>border</i>, <i>fontID</i>, <i>inputKind</i> : int) : int</pre>
Description	<p>Creates a text field and returns the text field's widget ID.</p> <p>A text field is used to create a line of text that can be edited by the user. The user can use the mouse to select part of the text and can enter text into the text field.</p> <p>If one or more text fields are enabled in a window, then one of the text fields will be active. This means that when any keystrokes are entered into the window, the active text field will receive the keystrokes. The active text field can be changed using the GUI.SetActive procedure.</p> <p>The <i>x</i> and <i>y</i> parameters specify the lower-left corner of the area in which the text will be drawn. The text field border is just outside the text drawing area. The <i>width</i> parameter specifies the width of the text drawing area. The height of the text field is determined by the height of the font used by the text field. The border of the text field is just outside the text drawing area, so GUI.GetWidth will return values slightly larger than <i>width</i>. The <i>actionProc</i> parameter specifies the name of the procedure to be called when the user presses ENTER (RETURN on a Macintosh) when the text field is active. The parameter is the current text in the text field.</p> <p>For GUI.CreateTextField, the border around the text field is always a line. For GUI.CreateTextFieldFull, the type of border is specified by the <i>border</i> parameter. The <i>border</i> parameter is one of 0, GUI.LINE, GUI.INDENT, or GUI.EXDENT. A border of 0 is the default and is the same as GUI.LINE. GUI.INDENT and GUI.EXDENT only display properly if the background colour has been set to <i>gray</i> using GUI.SetBackgroundColor. GUI.INDENT makes the text field appear indented or recessed. GUI.EXDENT makes the text field appear to stand out from the window. The <i>fontID</i> parameter specifies the font ID of the font to be used in the text field. The font ID is received from a <i>Font.New</i> call. Do not call <i>Font.Free</i> for this font ID until the label has been disposed of by calling GUI.Dispose. The <i>inputKind</i> parameter specifies the type of input accepted by the text field. This is one of 0, GUI.ANY, GUI.INT, or GUI.REAL. An input type of 0 is the default and is the same as GUI.ANY. GUI.ANY allows any type of input in the text</p>

field. *GUI.INTEGER* only allows positive integer input in the text field. *GUI.REAL* allows any real number input in the text field. Note that using *GUI.INTEGER* or *GUI.REAL* does not guarantee that the text field string can be converted to an integer or a real. The text could be a null string, and for *GUI.REAL* could be part of a number such as the string "-" or "1.25E" both of which are illegal numbers. (To check the conversion, use the *strintok* or *strrealok* functions before calling *strint* or *strreal*.)



Two Text Fields

Example The following program creates a text field and echoes it on the screen when the user presses ENTER.

```
import GUI in "%oot/lib/GUI"
View.Set ("graphics:200;100")

var nameTextField, addressTextField : int % The Text Field IDs.

procedure NameEntered (text : string)
    GUI.SetSelection (addressTextField, 0, 0)
    GUI.SetActive (addressTextField)
end NameEntered

procedure AddressEntered (text : string)
    GUI.SetSelection (nameTextField, 0, 0)
    GUI.SetActive (nameTextField)
end AddressEntered

GUI.SetBackgroundColor (gray)
var quitButton := GUI.CreateButton (52, 5, 100, "Quit", GUI.Quit)
nameTextField := GUI.CreateTextFieldFull (50, 70, 100, "",
    NameEntered, GUI.INDENT, 0, 0)
addressTextField := GUI.CreateTextFieldFull (50, 40, 100, "",
    AddressEntered, GUI.INDENT, 0, 0)
var nameLabel := GUI.CreateLabelFull (45, 70, "Name", 0, 0,
    GUI.RIGHT, 0)
var addressLabel := GUI.CreateLabelFull (45, 40, "Address", 0, 0,
    GUI.RIGHT, 0)
```

```

loop
    exit when GUI.ProcessEvent
end loop

GUI.Dispose (quitButton)
colorback (gray)
Text.Locate (maxrow - 1, 1)
put "Name = ", GUI.GetText (nameTextField)
    put "Address = ", GUI.GetText (addressTextField) ..

```

Details	<p>Only one text field is active at a time. The active text field has a blinking cursor (or its selection highlighted). If a keystroke occurs when a window has an active text field in it, the keystroke will be directed to the active text field. You can change which text field is active with the GUI.SetActive procedure or by simply clicking on another text field with the mouse.</p> <p>When multiple text fields are created in a window, the first text field created is active when the program begins.</p> <p>The current version of the text field does not support cut and paste or keyboard commands to extend the selection.</p> <p>Because strings are a maximum of 255 characters, this is the maximum number of characters in a text field.</p> <p>The TAB character cycles between different text fields in a window. It cycles through the text fields in the order in which they were created. BACK TAB (shift+TAB) cycles through the fields in reverse order.</p>
Details	<p>When GUI.CreateTextField or GUI.CreateTextFieldFull is called, the newly created picture will be displayed immediately unless GUI.DisplayWhenCreated has been called with the <i>display</i> parameter set to false.</p> <p>When a text field is not enabled, the text field cannot be made active and the text in the field cannot be edited.</p>
Details	<p>The following GUI subprograms can be called with a text field as the <i>widgetID</i> parameter:</p> <p style="padding-left: 40px;">GUI.Show, GUI.Hide, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.GetText, GUI.SetText, GUI.SetSelection, GUI.SetActive</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateTextField, not by calling CreateTextField.</p>
See also	<p>GUI.GetText and GUI.SetText for reading and setting the text in the text field. See also GUI.SetSelection for setting the selected area of the text. See also GUI.SetActive for making the text field active.</p>

GUI.CreateVerticalScrollBar[Full]



Syntax

```
GUI.CreateVerticalScrollBar ( x, y, size : int,  
                             min, max, start : int,  
                             actionProc : procedure x (value : int) ) : int
```

```
GUI.CreateVerticalScrollBarFull ( x, y, size : int,  
                                  min, max, start : int,  
                                  actionProc : procedure x (value : int),  
                                  arrowInc, pageInc, thumbSize : int) : int
```

Description

Creates a vertical (up-down) scroll bar and returns the scroll bar's widget ID.

A scroll bar is a widget that allows users to see a piece of a document that cannot be displayed on the screen in its entirety. The picture below shows a vertical scroll bar. To control a scroll bar, there are a few choices: the user can click on the thumb (the box in the scroll bar) and slide it up and down, or the user can click in the scroll bar itself above or below the thumb (in which case the thumb is moved up or down one "page"), or the user can click on the up or down arrows at the ends of the scroll bar (in which case the thumb is moved up or down one "arrow increment" or "line").



A Vertical Scroll Bar

The programmer defines a page or an arrow increment. When the value of the scroll bar changes, the *action procedure* of the scroll bar is called with the new value as a parameter. The *action procedure* should then redraw the contents using the new value of the scroll bar.

The range of values that the scroll bar will give is determined by the *min* and *max* parameters in the *Create* call. The left side of the scroll bar represents the minimum value, while the right represents the maximum value. There is also the "thumb size". This represents the range of values that can be seen at once on the screen.

By default, the arrow increment (the amount the value is changed when the scrolling arrows are pressed) is set to one. The page increment (the amount the value is changed when the user clicks in the bar to the right or left of the thumb) is set to one quarter the difference between the minimum and the maximum. The "thumb size" is set to zero (see the description of scroll bars for an explanation of the thumb size).

The *x* and *y* parameters specify the lower-left corner of the scroll bar. The *size* parameter specifies the length of the scroll bar (including the arrows) in pixels. The *min* and *max* parameters are the minimum and maximum values returned by the scroll bar. The *start* parameter is the initial value of the scroll bar and should be between *min* and *max* inclusive. The *actionProc* parameter is the name of a procedure that is called when the value of the scroll bar is changed. The parameter to the *action procedure* is the current value of the scroll bar.

Example The following program creates a vertical scroll bar. Whenever the scroll bar's value is changed, a message is displayed in the window.

```
import GUI in "%oot/lib/GUI"

View.Set ("graphics:85;200")
var scrollBar : int

procedure ScrollBarMoved (value : int)
    Text.Locate (9, 7)
    put "Scroll"
    Text.Locate (10, 8)
    put "Bar"
    Text.Locate (11, 8)
    put value : 3
end ScrollBarMoved

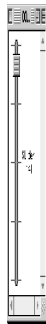
scrollBar := GUI.CreateVerticalScrollBar (10, 10, 180,
    50, 150, 50, ScrollBarMoved)
loop
    exit when GUI.ProcessEvent
end loop
```

Description	<p>For GUI.CreateVerticalScrollBarFull, the <i>arrowInc</i> parameter specifies the arrow increment (the amount the scroll bar's value is changed when the scroll arrows are pressed). The <i>pageInc</i> specifies the page increment (the amount the scroll bar's value is changed when the user clicks in the page left/right section of the scroll bar). The <i>thumbSize</i> parameter specifies the "thumb size". See the scroll bar explanation for more detail on a scroll bar's "thumb size".</p> <p>For example, if you have a window that can display 20 lines of text at once and there are 100 lines of text, you would set <i>min</i> to 1, <i>max</i> to 100 and <i>thumbSize</i> to 20. The value returned by the scroll bar would then be the line number of the first line on the screen to be displayed. When the scroll bar was at its maximum value, it would return 81, since by doing so, lines 81-100 would be displayed.</p>
Example	For an example program that scrolls a large picture over a smaller window, see GUI.CreateHorizontalScrollBar .
Details	<p>In some instances, you will want the the minimum and maximum values of the scroll bar to be reversed (right/top is minimum). In that case, call the GUI.SetSliderReverse procedure to flip the values of the scroll bar.</p> <p>Scroll bars always have a fixed height (for horizontal scroll bars) or width (for vertical scroll bars). To get the scroll bar's width, use the GUI.GetScrollBarWidth function.</p> <p>When GUI.CreateVerticalScrollBar or GUI.CreateVerticalScrollBarFull is called, the newly created scroll bar will be displayed immediately unless GUI.DisplayWhenCreated has been called with the <i>display</i> parameter set to false.</p> <p>When a scroll bar is not enabled, the gray in the bar is set to white and the thumb is not displayed. The scroll bar no longer responds to any mouse clicks until the scroll bar is enabled again.</p>
Details	<p>The following GUI subprograms can be called with a scroll bar as the <i>widgetID</i> parameter:</p> <p>GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.GetSliderValue, GUI.SetSliderValue, GUI.SetSliderMinMax, GUI.SetSliderSize, GUI.SetSliderReverse, GUI.SetScrollAmount</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateVerticalScrollBar, not by calling CreateVerticalScrollBar.</p>
See also	<p>GUI.GetSliderValue and GUI.SetSliderValue for reading and setting the value of a scroll bar, GUI.SetSliderMinMax for changing the minimum and maximum values of a scroll bar, and GUI.SetScrollAmount for changing the scrolling increments and thumb size of a scroll bar. See also GUI.SetSliderSize for setting the length of a scroll bar and GUI.SetSliderReverse for reversing the sense of a scroll bar.</p>

GUI.CreateVerticalSlider



Syntax	GUI.CreateVerticalSlider (<i>x</i> , <i>y</i> , <i>length</i> : int , <i>min</i> , <i>max</i> , <i>start</i> : int , <i>actionProc</i> : procedure <i>x</i> (<i>value</i> : int)) : int
Description	Creates a vertical (up-down) slider and returns the slider's widget ID. A slider is a widget that allows the user to set a continuous set of values. It has a real-life equivalent in things such as a stereo volume control.



A Vertical Slider

To control a slider, the user clicks on the slider box and drags it back and forth. Every time the value changes, a procedure is called with the new value as a parameter.

The range of values that the slider will give is determined by the *min* and *max* parameters in the *Create* call. The left side of the slider represents the minimum value, while the right represents the maximum value.

The *x* and *y* parameters specify the lower-left corner of the slider track. This means that the slider actually extends above and below this point (and slightly to the left of it to take into account the rounded end of the track). The *length* parameter specifies the length of the track in pixels. (You can use **GUI.GetX**, **GetY**, **GetWidth**, and **GetHeight** to get the exact dimensions of the slider.) The *min* and *max* parameters are the minimum and maximum values returned by the slider. The *start* parameter is the initial value of the slider and should be between *min* and *max* inclusive. The *actionProc* parameter is the name of a procedure that is called when the value of the slider is changed. The parameter to the *action procedure* is the current value of the slider.

Example The following program creates a vertical slider. Whenever the slider's value is changed, a message is displayed in the window.

```

import GUI in "%oot/lib/GUI"

View.Set ("graphics:85;200")
var slider : int

procedure SliderMoved (value : int)
    Text.Locate (9, 7)
    put "Slider"
    Text.Locate (10, 9)
    put value : 3
end SliderMoved

slider := GUI.CreateVerticalSlider (20, 10, 180,
    50, 150, 50, SliderMoved )
loop
    exit when GUI.ProcessEvent
end loop

```

Details	<p>In some instances, you will want the the minimum and maximum values of the slider to be reversed (right is minimum). In that case, call the GUI.SetSliderReverse procedure to flip the values of the slider.</p> <p>Sliders always have a fixed height (for horizontal sliders) or width (for vertical sliders).</p> <p>When GUI.CreateVerticalSlider or GUI.CreateVerticalSliderFull is called, the newly created slider will be displayed immediately unless GUI.DisplayWhenCreated has been called with the <i>display</i> parameter set to false.</p> <p>When a slider is not enabled, the appearance does not change. However, the slider no longer responds to any mouse clicks until it is enabled again.</p>
Details	<p>The following GUI subprograms can be called with a slider as the <i>widgetID</i> parameter:</p> <p style="text-align: center;"> GUI.Show, GUI.Hide, GUI.Enable, GUI.Disable, GUI.Dispose, GUI.GetX, GUI.GetY, GUI.GetWidth, GUI.GetHeight, GUI.SetPosition, GUI.SetSize, GUI.SetPositionAndSize, GUI.GetSliderValue, GUI.SetSliderValue, GUI.SetSliderMinMax, GUI.SetSliderSize, GUI.SetSliderReverse </p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.CreateVerticalSlider, not by calling CreateVerticalSlider.</p>
See also	<p>GUI.GetSliderValue and GUI.SetSliderValue for reading and setting the value of a slider, GUI.SetSliderMinMax for changing the minimum and maximum values of a slider. See also GUI.SetSliderSize for setting the length of a slider and GUI.SetSliderReverse for reversing the sense of a slider.</p>

GUI.Disable



Syntax	GUI.Disable (<i>widgetID</i> : int)
Description	<p>Disables a widget specified by <i>widgetID</i>.</p> <p>Used in conjunction with GUI.Enable to enable and disable widgets.</p> <p>Disabled widgets generally are "grayed out" to visually depict their disabled status.</p> <p>Disabled widgets do not respond to keystrokes or mouse clicks.</p>
Example	<p>The three color radio buttons are enabled only when the color check box is selected.</p> <pre>import GUI in "%oot/lib/GUI" View.Set ("graphics:100;100") var colorCheckBox, redRadio, greenRadio, blueRadio : int procedure DoNothing end DoNothing procedure ColorCheckBoxProc (filled : boolean) if filled then GUI.Enable (redRadio) GUI.Enable (greenRadio) GUI.Enable (blueRadio) else GUI.Disable (redRadio) GUI.Disable (greenRadio) GUI.Disable (blueRadio) end if end ColorCheckBoxProc colorCheckBox := GUI.CreateCheckBox (10, 80, "Use Color", ColorCheckBoxProc) redRadio := GUI.CreateRadioButton (33, 60, "Red", 0, DoNothing) greenRadio := GUI.CreateRadioButton (-1, -1, "Green", redRadio, DoNothing) blueRadio := GUI.CreateRadioButton (-1, -1, "Blue", greenRadio, DoNothing) ColorCheckBoxProc (false) loop exit when GUI.ProcessEvent end loop</pre>

Details	The following types of widgets can be enabled or disabled: Buttons, Check Boxes, Radio Buttons, Picture Buttons, Picture Radio Buttons, Horizontal Scroll Bars, Horizontal Sliders, Canvases, Text Fields, Vertical Scroll Bars, Vertical Sliders
Status	Exported qualified. This means that you can only call the procedure by calling GUI.Disable , not by calling Disable .
See also	GUI.Enable .

GUI.Dispose



Syntax	GUI.Dispose (<i>widgetID</i> : int)
Description	Eliminates the widget specified by <i>widgetID</i> . If the widget is visible, it is immediately made invisible before being deleted. It should be called in order to free up any memory that the widget might have allocated. Note that you cannot use the widget after it has been disposed of. If you wish to temporarily get rid of a widget, consider using the <i>Hide</i> method and then the <i>Show</i> method when you want to use it again.
Example	The following program waits for the Quit button to be pressed. When it is, the Quit button is deleted and a message is displayed in the center of the screen. <pre> import GUI in "%oot/lib/GUI" View.Set ("graphics:150;100") var button, message : int button := GUI.CreateButton (20, 40, 0, "Quit", GUI.Quit) loop exit when GUI.ProcessEvent end loop GUI.Dispose (button) message := GUI.CreateLabelFull (0, 0, "Done", maxx, maxy, GUI.CENTER + GUI.MIDDLE, 0) </pre>
Status	Exported qualified. This means that you can only call the procedure by calling GUI.Dispose , not by calling Dispose .

GUI.Draw...



Syntax

GUI.DrawArc (*widgetID*, *x*, *y*, *xRadius*, *yRadius* : **int**,
initialAngle, *finalAngle*, *Color* : **int**)

GUI.DrawBox (*widgetID*, *x1*, *y1*, *x2*, *y2*, *Color* : **int**)

GUI.DrawCls (*widgetID* : **int**)

GUI.DrawDot (*widgetID*, *x*, *y*, *Color* : **int**)

GUI.DrawFill (*widgetID*, *x*, *y* : **int**,
fillColor, *borderColor* : **int**)

GUI.DrawFillArc (*widgetID*, *x*, *y* : **int**,
xRadius, *yRadius* : **int**,
initialAngle, *finalAngle*, *Color* : **int**)

GUI.DrawFillBox (*widgetID*, *x1*, *y1*, *x2*, *y2* : **int**,
Color : **int**)

GUI.DrawFillMapleLeaf (*widgetID*, *x1*, *y1* : **int**,
x2, *y2*, *Color* : **int**)

GUI.DrawFillOval (*widgetID*, *x*, *y* : **int**,
xRadius, *yRadius* : **int**, *Color* : **int**)

GUI.DrawFillPolygon (*widgetID* : **int**,
x, *y* : **array** 1 .. * **of int**, *n* : **int**, *Color* : **int**)

GUI.DrawFillStar (*widgetID*, *x1*, *y1*, *x2*, *y2* : **int**,
Color : **int**)

GUI.DrawLine (*widgetID*, *x1*, *y1*, *x2*, *y2*, *Color* : **int**)

GUI.DrawMapleLeaf (*widgetID*, *x1*, *y1*, *x2*, *y2* : **int**,

```

        Color : int )
GUI.DrawOval ( widgetID, x, y : int,
               xRadius, yRadius, Color : int )
GUI.DrawPolygon ( widgetID : int,
                  x, y : array 1 .. * of int, n : int, Color : int )
GUI.DrawStar ( widgetID, x1, y1, x2, y2, Color : int )

GUI.DrawText ( widgetID : int, textStr : string,
               x, y : int, fontID, Color : int )

```

Description	<p>Performs a <i>Draw...</i> command to the canvas specified by <i>widgetID</i>.</p> <p>All of these routines are essentially the same as the similarly-named procedures in the <i>Draw</i> module. All coordinates are based on the canvas and all drawing is clipped to the canvas drawing surface. If the canvas is in "xor mode", all the drawing will be done with "xor" set. (See View.Set for more information about "xor".)</p> <p>The <i>widgetID</i> must specify a canvas widget.</p>
Example	See GUI.CreateCanvas for an example of GUI.DrawFillOval .
Status	<p>Exported qualified.</p> <p>This means that you can only call the procedures by calling GUI.Draw..., not by calling Draw...</p>
See also	GUI.CreateCanvas .

GUI.Enable



Syntax	GUI.Enable (<i>widgetID</i> : int)
Description	<p>Enables a disabled widget specified by <i>widgetID</i>.</p> <p>Used in conjunction with GUI.Disable to enable and disable widgets.</p> <p>Disabled widgets generally are "grayed out" to visually depict their disabled status.</p> <p>Disabled widgets do not respond to keystrokes or mouse clicks.</p>

Example	See GUI.Disable for an example of GUI.Enable .
Details	The following types of widgets can be enabled or disabled: Buttons, Check Boxes, Radio Buttons, Picture Buttons, Picture Radio Buttons, Horizontal Scroll Bars, Horizontal Sliders, Canvases, Text Fields, Vertical Scroll Bars, Vertical Sliders
Status	Exported qualified. This means that you can only call the procedure by calling GUI.Enable , not by calling Enable .
See also	GUI.Disable .

GUI.FontDraw



Syntax	GUI.FontDraw (<i>widgetID</i> : int , <i>textStr</i> : string , <i>x</i> , <i>y</i> , <i>fontID</i> , <i>Color</i> : int)
Description	Performs a <i>Font.Draw</i> command to the canvas specified by <i>widgetID</i> . This routine is essentially the same as the <i>Font.Draw</i> procedure in the <i>Font</i> module. All coordinates are based on the canvas and all drawing is clipped to the canvas drawing surface. If the canvas is in "xor mode", all the drawing will be done with "xor" set. (See View.Set for more information about "xor".) The <i>widgetID</i> must specify a canvas widget.
Status	Exported qualified. This means that you can only call the procedure by calling GUI.FontDraw , not by calling FontDraw .
See also	GUI.CreateCanvas .

GUI.GetCheckBox



Syntax	GUI.GetCheckBox (<i>widgetID</i> : int) : boolean
Description	Returns the status of the check box specified by <i>widgetID</i> . If the check box is set (has an X in it), <i>GetCheckBox</i> returns true, otherwise it returns false.
Example	See GUI.CreateCheckBox for an example of GUI.GetCheckBox .
Status	Exported qualified. This means that you can only call the function by calling GUI.GetCheckBox , not by calling GetCheckBox .
See also	GUI.CreateCheckBox .

GUI.GetEventTime



Syntax	GUI.GetEventTime : int
Description	Returns the time in milliseconds when the event (mouse button or keystroke) took place. This value is the same value as <i>Time.Elapsed</i> returns if called when the event was processed. This function should only be called in an <i>action procedure</i> or in a default mouse, keystroke, or null event handler, as it will return -1 when there is no event being processed. This event can be used as a timer for various functions such as determining whether a single click or a double click of the mouse button took place or for timing keyboard input.
Example	The following program times the interval between two button presses. <pre>import GUI in "%oot/lib/GUI" View.Set ("graphics:300;100") var startTime, startButton, finishButton : int procedure Start startTime := GUI.GetEventTime end Start</pre>

```

procedure Finish
  Text.Locate (1, 1)
  put "The time between button pressed is ",
    GUI.GetEventTime - startTime, " msec"
  GUI.Quit
end Finish

startButton := GUI.CreateButton (10, 10, 110, "Click First", Start)
finishButton := GUI.CreateButton (180, 10, 110, "Click Second", Finish)

loop
  exit when GUI.ProcessEvent
end loop

```

Status Exported qualified.
 This means that you can only call the function by calling
 GUI.GetEventTime , not by calling **GetEventTime** .

See also **GUI.ProcessEvent**.

GUI.GetEventWidgetID



Syntax **GUI.GetEventWidgetID : int**

Description Returns the widget ID of the widget that was activated by the mouse
 button press or the keystroke. This function should only be called in an
 action procedure, as it will return -1 when there is no event that activated a
 widget being processed.
 This function is used when a several buttons use the same action procedure
 to determine which button was pressed.

Example The following program prints a message stating which button was selected.

```

import GUI in "%oot/lib/GUI"
View.Set ("graphics:150;210")

var buttonNames : array 1 .. 5 of string := init ("Red", "Green",
  "Blue", "Yellow", "Purple")
var buttons : array 1 .. 5 of int

procedure ButtonPush
  for i : 1 .. 5
    if GUI.GetEventWidgetID = buttons (i) then
      Text.Locate (1, 1)

```

```

        put buttonNames (i), " selected"
    end if
end for
end ButtonPush
for i : 1 .. 5
    buttons (i) := GUI.CreateButton (10, 210 - 40 * i, 110,
        buttonNames (i), ButtonPush)
end for
loop
    exit when GUI.ProcessEvent
end loop

```

Status Exported qualified.
 This means that you can only call the function by calling
 GUI.GetEventWidgetID, not by calling **GetEventWidgetID**.

See also **GUI.ProcessEvent**.

GUI.GetEventWindow



Syntax **GUI.GetEventWindow : int**

Description Returns the window ID of the window in which the event (mouse button or
 keystroke) took place. This function should only be called in an *action*
 procedure or in a default mouse or keystroke event handler, as it will return
 -1 when there is no event being processed.

 This function is commonly used when several windows share the same
 layout. The same buttons in each window point to the same *action*
 procedures. To determine which button was actually pressed, the function is
 called to get the window.

Example The following program creates four windows in a row, each with a button
 that, when pressed, causes a star to be drawn in that window.

```

import GUI in "%oot/lib/GUI"

procedure DrawStar
    var windowID : int := GUI.GetEventWindow
    Window.Select (windowID)
    Draw.FillStar (25, 40, 175, 190, Rand.Int (10, 15))
end DrawStar

for i : 0 .. 3
    var window : int := Window.Open ("graphics:200;200")
    % Place window above task bar, across from previous one.

```

```

Window.SetPosition (window, 220 * i, 27)
var button : int := GUI.CreateButton (5, 5, 190, "Draw Star",
                                         DrawStar)
end for

loop
  exit when GUI.ProcessEvent
end loop

```

Status Exported qualified.
 This means that you can only call the function by calling **GUI.GetEventWindow**, not by calling **GetEventWindow**.

See also **GUI.ProcessEvent**.

GUI.GetHeight



Syntax **GUI.GetHeight** (*widgetID* : **int**) : **int**

Description Returns the actual height of a widget. Note that this may be different from the height specified in the *Create* call (especially since many widgets do not specify a height. The GUI module determines the actual height).

This function is used in conjunction with **GUI.GetX**, **GUI.GetY** and **GUI.GetWidth** to determine the bounds of a widget. The entire widget should always fit in the box (**GUI.GetX**, **GUI.GetY**) - (**GUI.GetX** + **GUI.GetWidth** - 1, **GUI.GetY** + **GUI.GetHeight** - 1)

The position and size of a widget is known only after it has been drawn to the screen. Attempting to get the location or dimesions of the widget may cause an uninitialized variable error.

Example The following procedure draws a red box around the widget specified by *widgetID*.

```

import GUI in "%oot/lib/GUI"

procedure BoxWidget (widgetID : int)
  var x, y, width, height : int
  x := GUI.GetX (widgetID)
  y := GUI.GetY (widgetID)
  width := GUI.GetWidth (widgetID)
  height := GUI.GetHeight (widgetID)
  Draw.Box (x - 1, x - 1, x + width, y + height, red)
  Draw.Box (x - 2, x - 2, x + width + 1, y + height + 1, red)
end BoxWidget

```

```
var title : int := GUI.CreateLabel (20, 20, "Frame This!")
BoxWidget (title)
```

- Status Exported qualified.
This means that you can only call the function by calling **GUI.GetHeight**, not by calling **GetHeight**.
- See also **GUI.GetX**, **GUI.GetY**, and **GUI.GetWidth**.

GUI.GetMenuBarHeight



- Syntax **GUI.GetMenuBarHeight : int**
- Description Returns the height of the menu bar. Useful when drawing or placing widgets to make certain that they don't overlap the menu bar.
- Example The following program draws a red box in the window just below the menu bar.

```
import GUI in "%oot/lib/GUI"

var menu : int := GUI.CreateMenu ("File")
var item : int := GUI.CreateMenuItem ("Quit", GUI.Quit)

Draw.FillBox (0, 0, maxx, maxy - GUI.GetMenuBarHeight - 2,
  brightred)

loop
  exit when GUI.ProcessEvent
end loop
```

- Status Exported qualified.
This means that you can only call the function by calling **GUI.GetMenuBarHeight**, not by calling **GetMenuBarHeight**.
- See also **GUI.CreateMenu**.

GUI.GetScrollBarWidth



Syntax	GUI.GetScrollBarWidth : int
Description	Returns the width of a scroll bar. Useful when placing a scroll bar widget beneath or beside another widget or object.
Example	See the <i>ScrollPic</i> program in GUI.CreateHorizontalScrollBarFull for an example of GUI.GetScrollBarWidth .
Status	Exported qualified. This means that you can only call the function by calling GUI.GetScrollBarWidth , not by calling GetScrollBarWidth .
See also	GUI.CreateHorizontalScrollBar and GUI.CreateVerticalScrollBar .

GUI.GetSliderValue



Syntax	GUI.GetSliderValue (<i>widgetID</i> : int) : int
Description	Returns the current value of a slider or scroll bar specified by <i>widgetID</i> . The <i>widgetID</i> must specify either scroll bar or a slider (horizontal or vertical).
Example	See the <i>ScrollPic</i> program in GUI.CreateHorizontalScrollBarFull for an example of GUI.GetSliderValue .
Status	Exported qualified. This means that you can only call the function by calling GUI.GetSliderValue , not by calling GetSliderValue .
See also	GUI.SetSliderValue for setting a slider or scroll bar's value. See also GUI.CreateHorizontalScrollBar , GUI.CreateVerticalScrollBar , GUI.CreateHorizontalSlider , and GUI.CreateVerticalSlider .

GUI.GetText



Syntax	GUI.GetText (<i>widgetID</i> : int) : string
Description	Returns the current text of a text field specified by <i>widgetID</i> . The <i>widgetID</i> must specify a text field widget.
Example	See GUI.CreateTextField for an example of GUI.GetText .
Status	Exported qualified. This means that you can only call the function by calling GUI.GetText , not by calling GetText .
See also	GUI.SetText for setting the text in a text field. See also GUI.CreateTextField .

GUI.GetVersion



Syntax	GUI.GetVersion : int
Description	Returns the current version of the GUI Procedure Library. Because the GUI Procedure Library is expected to grow, new versions will probably be made available at our web site http://www.holtsoft.com/turing . If you wish to use features that do not appear in earlier versions of the library, you can have your program check that the current available version meets the programs needs. GUI.GetVersion returns an integer from 100 - 999 and is read as 1.00 to 9.99.
Example	The following program fragment immediately exits if OOT does not support version 1.1 of the GUI Procedure Library <pre>import GUI in "%oot/lib/GUI" if GUI.GetVersion < 110 then put "You must update to at least version 1.1 of the" put "GUI Procedure Library to use this program." return end if</pre>

Details	In version 1.00 (shipped with MacOOT 1.5), GUI.GetVersion did not exist.
Status	Exported qualified. This means that you can only call the function by calling GUI.GetVersion , not by calling GetVersion .

GUI.GetWidth



Syntax	GUI.GetWidth (<i>widgetID</i> : int) : int
Description	<p>Returns the actual width of a widget. Note that this may be different from the width specified in the <i>Create</i> call (especially since some widgets do not specify a width. The GUI module determines the actual width).</p> <p>This function is used in conjunction with GUI.GetX, GUI.GetY and GUI.GetHeight to determine the bounds of a widget. The entire widget should always fit in the box (GUI.GetX, GUI.GetY) - (GUI.GetX + GUI.GetWidth - 1, GUI.GetY + GUI.GetHeight - 1)</p> <p>The position and size of a widget is known only after it has been drawn to the screen. Attempting to get the location or dimesions of the widget may cause an uninitialized variable error.</p>
Example	<p>The following procedure draws a red box around the widget specified by <i>widgetID</i>.</p> <pre> import GUI in "%oot/lib/GUI" procedure BoxWidget (widgetID : int) var x, y, width, height : int x := GUI.GetX (widgetID) y := GUI.GetY (widgetID) width := GUI.GetWidth (widgetID) height := GUI.GetHeight (widgetID) Draw.Box (x - 1, x - 1, x + width, y + height, red) Draw.Box (x - 2, x - 2, x + width + 1, y + height + 1, red) end BoxWidget var title : int := GUI.CreateLabel (20, 20, "Frame This!") BoxWidget (title) </pre>
Status	Exported qualified. This means that you can only call the function by calling GUI.GetWidth , not by calling GetWidth .
See also	GUI.GetX , GUI.GetY , and GUI.GetHeight .

GUI.Get{X,Y}



Syntax

GUI.GetX (*widgetID* : int) : int

GUI.GetY (*widgetID* : int) : int

Description

Returns the x coordinate of the left edge of a widget. Note that this may be different from the x coordinate specified in the widget's *Create* call. For example, if a radio button is created with right justification, the x coordinate in the *Create* method specifies the right edge while **GUI.GetX** will return the x coordinate of the left edge.

This function is used in conjunction with **GUI.GetWidth** and **GUI.GetHeight** to determine the bounds of a widget. The entire widget should always fit in the box (GUI.GetX, GUI.GetY) - (GUI.GetX + GUI.GetWidth - 1, GUI.GetY + GUI.GetHeight - 1)

The position and size of a widget is known only after it has been drawn to the screen. Attempting to get the location or dimensions of the widget may cause an uninitialized variable error.

Example

The following procedure draws a red box around the widget specified by *widgetID*.

```
import GUI in "%oot/lib/GUI"
procedure BoxWidget (widgetID : int)
  var x, y, width, height : int
  x := GUI.GetX (widgetID)
  y := GUI.GetY (widgetID)
  width := GUI.GetWidth (widgetID)
  height := GUI.GetHeight (widgetID)
  Draw.Box (x - 1, x - 1, x + width, y + height, red)
  Draw.Box (x - 2, x - 2, x + width + 1, y + height + 1, red)
end BoxWidget

var title : int := GUI.CreateLabel (20, 20, "Frame This!")
BoxWidget (title)
```

Status

Exported qualified.

This means that you can only call the function by calling **GUI.GetX**, not by calling **GetX**.

See also

GUI.GetHeight and **GUI.GetWidth**.

GUI.Hide



Syntax	GUI.Hide (<i>widgetID</i> : int)
Description	<p>Hides a widget specified by <i>widgetID</i>. Used in conjunction with <i>Show</i> to hide and show widgets. Hidden widgets cannot get events (i.e. respond to keystrokes or mouse clicks). If an active text field (see text field) is hidden, then any keystrokes in the window will be ignored.</p> <p>In most cases where a widget is to appear, then disappear, then appear again, it is advised to create the widget once and hide it until it is to appear, whereupon GUI.Show is called. When the user is finished with the widget, the widget is hidden using GUI.Hide. This saves the overhead of creating and disposing of the same widget several times.</p>
Example	See GUI.SetDisplayWhenCreated for an example of GUI.Hide .
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.Hide, not by calling Hide.</p>
See also	GUI.Show .

GUI.HideMenuBar



Syntax	GUI.HideMenuBar
Description	<p>Hides the menu bar in the selected window. No menu items can be selected when the menu bar is hidden. (Menu item shortcuts are ignored while the menu bar is hidden.)</p>
Example	See GUI.SetMouseEventHandler for an example of GUI.HideMenuBar .
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.HideMenuBar, not by calling HideMenuBar.</p>

See also

GUI.ShowMenuBar. See also **GUI.CreateMenu.**

GUI.OpenFile



Syntax

GUI.OpenFile (*title* : **string**) : **string**

Description

Displays an "Open File" dialog box to obtain the name of an already existing file. The caption (a window title under MS Windows, a string in a Macintosh dialog box) is specified by the *title* parameter. The function uses a dialog box specific to the operating system the program is being run on.

If the user did not choose a file (i.e. hit the *Cancel* button in the dialog), the function returns "" (the empty string).

Note: This function is **not** available in the current version of the GUI Procedure Library (shipping with WinOOT 3.1, DOS OOT 2.5 and MacOOT 1.5). It is documented here for use with future shipping version of Object Oriented Turing. It is likely to be implemented in the version of Object Oriented Turing released in September 2000. Check the release notes that are found in the on-line help to find out if this function is now available.

Example

The following program asks the user for the name of a file and then echoes the contents of it.

```
import GUI in "%oot/lib/GUI"

var fileName, line : string
var streamNumber : int

fileName := GUI.OpenFile ("Choose a Text File")

open : streamNumber, fileName, get
assert streamNumber > 0
loop
  exit when eof (streamNumber)
  get : streamNumber, line : *
  put line
end loop

close : streamNumber
```

Status

Exported qualified.

This means that you can only call the function by calling **GUI.OpenFile**, not by calling **OpenFile**.

GUI.OpenFileFull



Syntax	GUI.OpenFileFull (<i>title</i> , <i>filter</i> : string , <i>startDir</i> : string) : string
Description	<p>Displays an "Open File" dialog box to obtain the name of an already existing file. The caption (a window title under MS Windows, a string in a Macintosh dialog box) is specified by the <i>title</i> parameter. The list of files shown is specified by the <i>filter</i> parameter. The initial directory to be displayed is specified by the <i>startDir</i> parameter. The function uses a dialog box specific to the operating system the program is being run on.</p> <p>The <i>filter</i> parameter is a file name suffix that should be displayed. Multiple suffixes can be specified by separating them with commas. If the user specifies the empty string for <i>filter</i>, then all the files in the directory are displayed. If the <i>startDir</i> parameter is empty, or if it specifies a non-existent directory, then the current directory is displayed in the "Open File" dialog box.</p> <p>If the user did not choose a file (i.e. hit the <i>Cancel</i> button in the dialog), the function returns "" (the empty string).</p> <p>Note: This function is not available in the current version of the GUI Procedure Library (shipping with WinOOT 3.1, DOS OOT 2.5 and MacOOT 1.5). It is documented here for use with future shipping version of Object Oriented Turing. It is likely to be implemented in the version of Object Oriented Turing released in September 2000. Check the release notes that are found in the on-line help to find out if this function is now available.</p>
Example	<p>The following program asks the user to select a file ending in ".txt". The initial directory is the root directory of the C drive.</p> <pre>import GUI in "%oot/lib/GUI" var fileName : string fileName := GUI.OpenFileFull ("Choose a Text File", "txt", "C:\\")</pre>
Details	<p>If a suffix is placed in single quotes, it will be ignored on all but the Apple Macintosh, where it will specify a Macintosh file type.</p>
Example	<p>The example makes the dialog box display all files ending in ".txt" or ".text" on all systems but the Macintosh. On the Apple Macintosh, only files of file type 'TEXT' will be displayed.</p> <pre>fileName := GUI.OpenFileFull ("Open", "txt,text,'TEXT'", "")</pre>

Status Exported qualified.
This means that you can only call the function by calling **GUI.OpenFileFull**, not by calling **OpenFileFull**.

GUI.Pic...



Syntax **GUI.PicDraw** (*widgetID* : **int**, *picID*, *x*, *y*, *mode* : **int**)

GUI.PicNew (*widgetID* : **int**, *x1*, *y1*, *x2*, *y2* : **int**) : **int**

GUI.PicScreenLoad (*widgetID* : **int**, *fileName* : **string**,
 x, *y*, *mode* : **int**)

GUI.PicScreenSave (*widgetID* : **int**, *x1*, *y1*, *x2*, *y2* : **int**,
 fileName : **string**)

Description Performs a *Pic...* command to the canvas specified by *widgetID*.
All of these routines are essentially the same as the similarly-named
procedures in the *Pic* module. All coordinates are based on the canvas and
all drawing is clipped to the canvas drawing surface.

Example See the *ScrollPic* program in **GUI.CreateHorizontalScrollBarFull** for an
example of **GUI.PicDraw**.

Status Exported qualified.
This means that you can only call the procedures by calling **GUI.Pic...**, not
by calling **Pic...**.

See also **GUI.CreateCanvas**.

GUI.ProcessEvent



Syntax **GUI.ProcessEvent** : **boolean**

Description	<p>This function processes a single event (a mouse button press or a keystroke). If the event activates a widget, then the <i>action procedure</i> of the widget is called.</p> <p>The function returns false until GUI.Quit is called. It then returns true.</p> <p>In order for the widgets to function once placed, the GUI.ProcessEvent must be called continually. Without a call to GUI.ProcessEvent, the widgets will appear, but will not react to mouse clicks or keystrokes.</p> <p>Almost all programs involving the GUI Procedure Library have the following code fragment in the program. This code fragment is often called the <i>event loop</i>.</p> <pre> loop exit when GUI.ProcessEvent end loop </pre> <p>The loop runs continuously until GUI.Quit is called, whereupon GUI.ProcessEvent will return true and the loop will exit. The rest of the program is reached through the <i>action procedures</i> that are called when the user interacts with various widgets.</p>
Details	<p>It is usually desirable to allow the user some way of quitting the program without having to abort it. This can be done most simply by adding a Quit button and placing it in an appropriate location.</p>
Example	<p>Here is program that does nothing but wait for the user to press the quit button.</p> <pre> import GUI in "%oot/lib/GUI" var quitButton : int := GUI.CreateButton (10, 10, 0, "Quit", GUI.Quit) loop exit when GUI.ProcessEvent end loop </pre>
Details	<p>To find out which widget was activated and called the <i>action procedure</i> (necessary if several widgets have the same <i>action procedure</i>), you can call GUI.GetEventWidgetID. To get the exact time that the event occurred, you can call GUI.GetEventTime. To get the window in which the event took place, you can call GUI.GetEventWindow.</p>
If a mouse click	<p>occured, but did not activate any widget, then the default mouse event handler is called. By default, this does nothing. However, if you want your program to respond to mouse events that do not affect a widget, call GUI.SetMouseEventHandler to specify your own default mouse event handler.</p> <p>If a keystroke occurred, but did not activate any widget (i.e. it wasn't a short cut for a widget and there are no text fields in the window) then the default keystroke handler is called. By default, this does nothing. However, if you want your program to respond to keystroke events that do not affect a widget, call GUI.SetKeyEventHandler to specify your own default key event handler.</p>

If no event occurred, then the null event handler is called. By default, this does nothing. However, if you want your program to perform some action repetetively when it is not doing anything else, then call **GUI.SetNullEventHandler** to specify your own null event handler. The null event handler is often used for such things as updating a clock and making certain that music is playing in the background.

Status Exported qualified.

This means that you can only call the procedures by calling **GUI.PProcessEvent**, not by calling **ProcessEvent**.

See also **GUI.GetEventWidgetID**, **GUI.GetEventTime**, and **GUI.GetEventWindow** for obtaining information about an event in an *action procedure*. See also **GUI.SetMouseEventHandler**, **GUI.SetKeyEventHandler** and **GUI.SetNullEventHandler** for handling mouse, keyboard and null events in the *event loop*. See also **GUI.Quit** for information on exiting the *event loop*.

GUI.Quit



Syntax **GUI.Quit**

Description This procedure causes **GUI.ProcessEvent** to return **true**. If the program is structured properly with a

```
loop
  exit when GUI.ProcessEvent
end loop
```

at the end of the program, then the program will exit the loop after finishing the current *action procedure*. This procedure is usually called from the *action procedure* of a Quit button or Exit menu item.

Example Here is program that does nothing but wait for the user to press the quit button or type the letter 'Q', 'q', 'X', or 'x'.

```
import GUI in "%oot/lib/GUI"

procedure KeyHandler (ch : char)
  if ch = 'Q' or ch = 'q' or ch = 'X' or ch = 'x' then
    GUI.Quit
  end if
end KeyHandler

var quitButton : int := GUI.CreateButton (10, 10, 0, "Quit", GUI.Quit)
GUI.SetKeyEventHandler (KeyHandler)
```



```

loop
  exit when GUI.ProcessEvent
end loop

```

```

put "Done!"

```

Status	Exported qualified. This means that you can only call the procedures by calling GUI.Quit , not by calling Quit .
See also	GUI.ProcessEvent .

GUI.Refresh



Syntax	GUI.Refresh
Description	<p>This routine redraws all the widgets in the currently-selected window. This is used when some form of drawing may have overwritten the widgets in a window.</p> <p>It is often used when there is some possibility that the widgets may have been drawn over. For example, a program that places buttons on top of a background image should call GUI.Refresh when the image is changed.</p>
Details	GUI.Refresh is used by the GUI Library to redraw all the widgets when the background colour of a window has changed.
Status	<p>Exported qualified.</p> <p>This means that you can only call the procedures by calling GUI.Refresh, not by calling Refresh.</p>

GUI.SaveFile



Syntax	GUI.SaveFile (<i>title</i> : string) : string
--------	--

Description	<p>Displays an "Save File" dialog box to obtain the name of a file. The caption (a window title under MS Windows, a string in a Macintosh dialog box) is specified by the <i>title</i> parameter. The function uses a dialog box specific to the operating system the program is being run on.</p> <p>If the user did not choose a file (i.e. hit the <i>Cancel</i> button in the dialog), the function returns "" (the empty string).</p> <p>Note: This function is not available in the current version of the GUI Procedure Library (shipping with WinOOT 3.1, DOS OOT 2.5 and MacOOT 1.5). It is documented here for use with future shipping version of Object Oriented Turing. It is likely to be implemented in the version of Object Oriented Turing released in September 2000. Check the release notes that are found in the on-line help to find out if this function is now available.</p>
Example	<p>The following program asks the user for the name of a file and then writes the numbers 1 to 10 in it.</p> <pre> import GUI in "%oot/lib/GUI" var fileName : string var streamNumber : int fileName := GUI.SaveFile ("Choose a Text File") open : streamNumber, fileName, put assert streamNumber > 0 for i : 1 .. 10 put : streamNumber, i end loop close : streamNumber </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.SaveFile, not by calling SaveFile.</p>

GUI.SaveFileFull



Syntax	<pre> GUI.SaveFileFull (<i>title</i>, <i>filter</i> : string, <i>startDir</i> : string) : string </pre>
--------	---

Description	<p>Displays an "Save File" dialog box to obtain the name of an already existing file. The caption (a window title under MS Windows, a string in a Macintosh dialog box) is specified by the <i>title</i> parameter. The list of files shown is specified by the <i>filter</i> parameter. The initial directory to be displayed is specified by the <i>startDir</i> parameter. The function uses a dialog box specific to the operating system the program is being run on.</p> <p>The <i>filter</i> parameter is a file name suffix that should be displayed. Multiple suffixes can be specified by separating them with commas. If the user specifies the empty string for <i>filter</i>, then all the files in the directory are displayed. If the <i>startDir</i> parameter is empty, or if it specifies a non-existent directory, then the current directory is displayed in the "Open File" dialog box.</p> <p>If the user did not choose a file (i.e. hit the <i>Cancel</i> button in the dialog), the function returns "" (the empty string).</p> <p>Note: This function is not available in the current version of the GUI Procedure Library (shipping with WinOOT 3.1, DOS OOT 2.5 and MacOOT 1.5). It is documented here for use with future shipping version of Object Oriented Turing. It is likely to be implemented in the version of Object Oriented Turing released in September 2000. Check the release notes that are found in the on-line help to find out if this function is now available.</p>
Example	<p>The following program segment asks the user for the name of a file ending in ".txt". The initial directory is the root directory of the C drive.</p> <pre>var fileName : string := GUI.SaveFileFull ("Choose a Text File", "txt", "C:\\")</pre>
Details	<p>If a suffix is placed in single quotes, it will be ignored on all but the Apple Macintosh, where it will specify a Macintosh file type.</p>
Example	<p>The following program segment asks the user for the name of a file. It displays files of type 'TEXT'. The initial directory is the "Turing Programs" directory on the "Macintosh HD" volume.</p> <pre>var fileName : string := GUI.SaveFileFull ("Choose a Text File", "'TEXT'", "Macintosh HD:Turing Programs")</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.SaveFileFull, not by calling SaveFileFull.</p>

GUI.SelectRadio



Syntax	GUI.SelectRadio (<i>widgetID</i> : int)
Description	Selects a radio button specified by <i>widgetID</i> . The previously-selected radio button is "de-selected". The <i>action procedure</i> of the radio button is called.
Example	<p>The following program creates six radio buttons. Selecting one of the buttons cause the bottom two radio buttons to become selected.</p> <pre> import GUI in "%oot/lib/GUI" View.Set ("graphics:350;110") var radio : array 1 .. 6 of int % The radio button IDs. procedure RadioPressed Text.Locate (1, 1) for i : 1 .. 6 if radio (i) = GUI.GetEventWidgetID then put "Radio Button " , i, " Selected" end if end for end RadioPressed procedure Select GUI.SelectRadio (radio (3)) GUI.SelectRadio (radio (6)) end Select radio (1) := GUI.CreateRadioButton (15, maxy - 35, "Radio Button 1", 0, RadioPressed) radio (2) := GUI.CreateRadioButton (-1, -1, "Radio Button 2", radio (1), RadioPressed) radio (3) := GUI.CreateRadioButton (-1, -1, "Radio Button 3", radio (2), RadioPressed) radio (4) := GUI.CreateRadioButtonFull (maxx - 15, maxy - 35, "Radio Button 4", 0, RadioPressed, GUI.RIGHT, GUI.NONE) radio (5) := GUI.CreateRadioButtonFull (-1, -1, "Radio Button 5", radio (4), RadioPressed, GUI.RIGHT, GUI.NONE) radio (6) := GUI.CreateRadioButtonFull (-1, -1, "Radio Button 6", radio (5), RadioPressed, GUI.RIGHT, GUI.NONE) var selectButton : int := GUI.CreateButton (15, 10, 100, "Select Bottom Buttons", Select) var quitButton : int := GUI.CreateButton (maxx - 15 - 100, 10, 100, "Quit", GUI.Quit) loop exit when GUI.ProcessEvent end loop </pre>

Status	Exported qualified. This means that you can only call the procedures by calling GUI.SelectRadio , not by calling SelectRadio .
See also	GUI.CreateRadioButton and GUI.CreatePictureRadioButton .

Win DOS Mac ●●● ○×× X TOOT Term GUI.SetActive

Syntax	GUI.SetActive (<i>widgetID</i> : int)
Description	Makes a text field specified by <i>widgetID</i> active. If the text field is not in an active window, then the text field will become active when the window does. If another text field was active in the window, it is deactivated.
Example	See GUI.CreateTextField for an example of GUI.SetActive .
Status	Exported qualified. This means that you can only call the procedures by calling GUI.SetActive , not by calling SetActive .
See also	GUI.CreateTextField .

Win DOS Mac ●●● ○×× X TOOT Term GUI.SetBackgroundColor

Syntax	GUI.SetBackgroundColor (<i>Color</i> : int)
Description	<p>Changes the background colour of the currently-selected window to the color specified by <i>Color</i>. This does <i>not</i> change the value of color 0 in the window. Instead it fills the entire window with the new background color and then redraws all the widgets.</p> <p>For indented and extended items, the background color is assumed to be set to <i>gray</i>.</p> <p>The alternate spelling is GUI.SetBackgroundColour</p>

Example See **GUI.CreateFrame** for an example of **GUI.SetBackgroundColour**.

Status Exported qualified.
This means that you can only call the procedures by calling **GUI.SetBackgroundColor**, not by calling **SetBackgroundColor**.

GUI.SetCheckBox



Syntax **GUI.SetCheckBox** (*widgetID* : **int**, *status* : **boolean**)

Description Sets the status of a check box specified by *widgetID*. If *status* is **true**, the check box is filled (marked with an 'X'). If *status* is **false**, the check box is set empty. **GUI.SetCheckBox** calls the check box's *action procedure* with the new status and redraws the widget with the new status.

Example See **GUI.CreateCheckBox** for an example of **GUI.SetCheckBox**.

Status Exported qualified.
This means that you can only call the procedures by calling **GUI.SetCheckBox**, not by calling **SetCheckBox**.

See also **GUI.CreateCheckBox**.

GUI.SetDefault



Syntax **GUI.SetDefault** (*widgetID* : **int**, *default* : **boolean**)

Description Sets the "default status" of a button specified by *widgetID*. If a button is the default button, then it is drawn with a heavy outline and it is activated when the user presses ENTER (RETURN on a Macintosh).
Only one button can be the default button per window. If a button is set to be the default button, then the previous default button has its "default status" removed.

Example	See GUI.CreateTextField for an example of GUI.SetCheckBox .
Status	Exported qualified. This means that you can only call the procedures by calling GUI.SetDefault , not by calling SetDefault .
See also	GUI.CreateButton .

GUI.SetDisplayWhenCreated



Syntax	GUI.SetDisplayWhenCreated (<i>display</i> : boolean)
Description	<p>By default, whenever a widget is created with a <i>GUI.Create...</i> procedure, the widget instantly appears. Sometimes, this is not the desired behaviour. For example, if several widgets are to occupy the same location with only one being displayed at a time, then it is desirable not to have the widget appear when first created.</p> <p>If a widget is not displayed when created, then <i>GUI.Show</i> must be called to make the widget visible.</p> <p>If the <i>display</i> parameter is true, then widgets are displayed immediately upon creation. If the <i>display</i> parameter is set to false, then the widget is not made visible on creation and <i>GUI.Show</i> must be called to display the widget.</p>
Example	<p>The following program toggles the visibility of the frame when the button is pushed. The frame starts out invisible.</p> <pre> import GUI in "%oot/lib/GUI" View.Set ("graphics:150;100") var visible : boolean := false var button, frame : int procedure Toggle if visible then GUI.Hide (frame) else GUI.Show (frame) end if visible := not visible end Toggle button := GUI.CreateButton (25, 40, 0, "Toggle Frame", Toggle) GUI.SetDisplayWhenCreated (false) </pre>


```

var quitButton : int := GUI.CreateButtonFull (115, 5, 100, "Quit",
      GUI.Quit, 0, 'Q', false)
loop
  exit when GUI.ProcessEvent
end loop

```

Status Exported qualified.
 This means that you can only call the function by calling
 GUI.SetKeyEventHandler, not by calling **SetKeyEventHandler**.

See also **GUI.ProcessEvent**.

GUI.SetLabel



Syntax **GUI.SetLabel** (*widgetID* : **int**, *text* : **string**)

Description Changes the text of a widget specified by *widgetID* to *text*. This procedure
 can accept a button, check box, radio button, label, or a labelled frame
 widget as the *widgetID* parameter.

 In most cases, if the text will not fit in the widget's current size, the widget
 will be resized to fit the text. If the widget was made larger to fit the text
 and then the text is changed, the widget will be resized as appropriate for
 the original *width* specified and the new text.

Example The following program changes the text in the button whenever a
 keystroke occurs. When the text is changed back to "Quit", the button
 assumes a width of 100 again.

```

import GUI in "%oot/lib/GUI"
View.Set ("graphics:220;50")

var short : boolean := true
var button : int

procedure KeyHandler (ch : char)
  if short then
    GUI.SetLabel (button, "Press This Button to Quit")
  else
    GUI.SetLabel (button, "Quit")
  end if
  short := not short
end KeyHandler

GUI.SetKeyEventHandler (KeyHandler)
button := GUI.CreateButton (10, 5, 100, "Quit", GUI.Quit)

```

	<pre> loop exit when GUI.ProcessEvent end loop </pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.SetLabel, not by calling SetLabel.</p>
See also	GUI.CreateButton , GUI.CreateCheckBox , GUI.CreateRadioButton , GUI.CreateLabel , and GUI.CreateLabelledFrame .

GUI.SetMouseEventHandler



Syntax	GUI.SetMouseEventHandler (<i>mouseEventHandler</i> : procedure <i>x</i> (<i>mx</i> , <i>my</i> : int))
Description	<p>Sets the new default mouse event handler. The <i>mouseEventHandler</i> parameter is the name of a procedure that is called every time GUI.ProcessEvent is called and there is a mouse button down which is not handled by any widget. The <i>mx</i> and <i>my</i> parameters in the <i>mouseEventHandler</i> are the location of mouse when the button was pressed.</p> <p>This procedure is used by programs to allow for mouse input in a program that uses widgets.</p>
Example	<p>This is a program that allows the user to place stars on the screen. The menu bar allows the user to quit the program at any time. The user can also toggle the appearance of the menu bar by pressing any key.</p>

```

import GUI in "%oot/lib/GUI"

var starX, starY, starColor : array 1 .. 100 of int
var numStars : int := 0
var menuVisible : boolean := true

procedure DrawStar (i : int)
  if menuVisible then
    View.ClipSet (0, 0, maxx,
                  maxy - GUI.GetMenuBarHeight)
  end if
  Draw.FillStar (starX (i) - 20, starY (i) - 20, starX (i) + 20,
                 starY (i) + 20, starColor (i))
  View.ClipOff
end DrawStar

procedure Redraw

```

```

    for i : 1 .. numStars
        DrawStar (i)
    end for
    Text.Locate (maxrow, 1)
    put "Press any key to toggle menu bar" ..
end Redraw

procedure KeyHandler (ch : char)
    if menuVisible then
        GUI.HideMenuBar
    else
        GUI.ShowMenuBar
    end if
    menuVisible := not menuVisible
    Redraw
end KeyHandler

procedure MouseHandler (x, y : int)
    if numStars = 100 then
        Text.Locate (maxrow, 1)
        put "Maximum number of stars exceeded!" ..
        return
    end if
    numStars += 1
    starX (numStars) := x
    starY (numStars) := y
    starColor (numStars) := Rand.Int (9, 15)
    DrawStar (numStars)
end MouseHandler

var menu : int := GUI.CreateMenu ("File")
var menuItem : int := GUI.CreateMenuItemFull ("Quit",
    GUI.Quit, '^Q', false)
GUI.SetKeyEventHandler (KeyHandler)
GUI.SetMouseEventHandler (MouseHandler)
Redraw
loop
    exit when GUI.ProcessEvent
end loop

```

Status	Exported qualified. This means that you can only call the function by calling GUI.SetMouseEventHandler , not by calling SetMouseEventHandler .
See also	GUI.ProcessEvent .

GUI.SetNullEventHandler



Syntax	GUI.SetNullEventHandler (<i>nullHandler</i> : procedure <i>x</i> ())
Description	<p>Sets the new null event handler. The <i>nullHandler</i> parameter is the name of a procedure that is called every time GUI.ProcessEvent is called and there are no mouse button presses or keystrokes to be processed.</p> <p>This is used by programs that need to call subprograms often, but do not wish to interrupt the action of user widgets.</p>
Example	<p>The following program has a Quit button. When no widgets are being processed, a clock in the corner is updated.</p> <pre>import GUI in "%oot/lib/GUI" View.Set ("graphics:220;50") var oldTime : string := "" var button : int procedure NullHandler var newTime : string := Time.Date newTime := newTime (11 .. *) if newTime not= oldTime then Text.Locate (maxrow, maxcol - 9) put newTime .. oldTime := newTime end if end NullHandler GUI.SetNullEventHandler (NullHandler) button := GUI.CreateButton (10, 5, 100, "Quit", GUI.Quit) loop exit when GUI.ProcessEvent end loop</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.SetNullEventHandler, not by calling SetNullEventHandler.</p>
See also	GUI.ProcessEvent .

GUI.SetPosition



Syntax	GUI.SetPosition (<i>widgetID</i> , <i>x</i> , <i>y</i> : int)
Description	Moves a widget specified by <i>widgetID</i> to the location (<i>x</i> , <i>y</i>). If the widget is visible, it is moved immediately to the new location. If the widget is hidden, it will appear at the new location when the <i>Show</i> procedure is called. Note that the <i>x</i> and <i>y</i> specified here are the same as in the <i>Create</i> method. For example, if you had specified a check box to be right justified in the <i>CreateCheckBoxFull</i> function, then (<i>x</i> , <i>y</i>) in a call to <i>SetPosition</i> would specify the lower-right corner as opposed to the lower-left corner.
Example	<p>The following program moves the button every time the button is pressed.</p> <pre>import GUI in "%oot/lib/GUI" var button : int procedure MoveButton var newX, newY : int newX := Rand.Int (0, maxx - GUI.GetWidth (button)) newY := Rand.Int (0, maxy - GUI.GetHeight (button)) GUI.SetPosition (button, newX, newY) end MoveButton button := GUI.CreateButton (100, 100, 0, "Move Button", MoveButton) loop exit when GUI.ProcessEvent end loop</pre>
Status	Exported qualified. This means that you can only call the function by calling GUI.SetPosition , not by calling SetPosition .

GUI.SetPositionAndSize



Syntax	GUI.SetPositionAndSize (<i>widgetID</i> , <i>x</i> , <i>y</i> : int ,
--------	--

width, height : int)

Description Changes the position and size of the widget specified by *widgetID* simultaneously. The *x*, *y*, *width* and *height* parameters have the same meaning as in the **GUI.Create** function for that widget. Any widget except a menu or a menu item can be resized, although for some widgets, the *width* or *height* parameter may be ignored.

GUI.SetPositionAndSize works the same way as the **GUI.SetPosition** and **GUI.SetSize** procedures.

Example The following program moves and resizes the button every time the button is pressed.

```
import GUI in "%oot/lib/GUI"

var button, minWidth, minHeight : int

procedure MoveButton
  var newX, newY, newWidth, newHeight : int
  newWidth := max (minWidth, Rand.Int (0, 200))
  newHeight := max (minHeight, Rand.Int (0, 100))
  newX := Rand.Int (0, maxx - newWidth)
  newY := Rand.Int (0, maxy - newHeight)
  GUI.SetPositionAndSize (button, newX, newY,
    newWidth, newHeight)
end MoveButton

button := GUI.CreateButton (100, 100, 0, "Move Button",
  MoveButton)
minHeight := GUI.GetHeight (button)
minWidth := GUI.GetWidth (button)
loop
  exit when GUI.ProcessEvent
end loop
```

Status Exported qualified.

This means that you can only call the function by calling **GUI.SetPositionAndSize**, not by calling **SetPositionAndSize**.

GUI.SetScrollAmount

Win DOS Mac

 X TOOT Term

Syntax **GUI.SetScrollAmount** (*widgetID : int*,
arrowInc, pageInc, thumbSize : int)

Description Sets a scroll bar's arrow increment, page increment and thumb size. Redraws the scroll bar to take into account the new thumb size. The *widgetID* specifies the scroll bar to be changed. The *arrowInc* parameter is the new arrow increment (the amount the scroll bar's value is changed when the scroll arrows are pressed). A value of -1 means to use the previously-defined arrow increment value. The *pageInc* parameter specifies the new page increment (the amount the scroll bar's value is changed when the user clicks in the page up/down section of the scroll bar). A value of -1 means to use the previously-defined page increment value. The *thumbSize* parameter specifies the new thumb size. See the scroll bar explanation for more detail on a scroll bar's thumb size. A value of -1 means to use the previously-defined thumb size.

Example The following program displays an image in a canvas in a window. If the image is larger than the canvas, scroll bars to the bottom and left are used to allow the user to see the entire image. A text field allows users to load additional images whenever the "Load File" button is pressed.

```
% The "ScrollPic2" program.
import GUI in "%oot/lib/GUI"

% The maximum width/height of the canvas.
const maxSize : int := 220
const leftBorder : int := 15 % Left margin.
const bottomBorder : int := 25 % Bottom margin.

var h, v : int % The scroll bars.
var canvas : int % The canvas.
var pic : int % The picture.
var fileNameField : int % The file name text field.
var errorLabel : int % The error message label.
var loadButton : int % The "Load Picture" button.

procedure ScrollPic (ignore : int)
    % Get the current value of the scroll bars.
    var x : int := GUI.GetSliderValue (h)
    var y : int := GUI.GetSliderValue (v)
    GUI.PicDraw (canvas, pic, -x, -y, picCopy)
end ScrollPic
```

```
procedure LoadFile (fileName : string)
    var picWidth, picHeight, canvasWidth, canvasHeight : int
    var newPic : int := Pic.FileNew (fileName)
    if newPic <= 0 then
        GUI.SetLabel (errorLabel,
            "Error loading picture: " + Error.LastMsg)
        GUI.SetSelection (fileNameField, -1, -1)
        return
    else
        GUI.SetLabel (errorLabel, "")
        pic := newPic
    end if
    picWidth := Pic.GetWidth (pic)
    picHeight := Pic.GetHeight (pic)
    canvasWidth := min (picWidth, maxSize)
```

```

    canvasHeight := min (picHeight, maxSize)
    % Hide the canvas and the three items, readjust them
    % and then show them.
    GUI.Hide (canvas)
    GUI.Hide (h)
    GUI.Hide (v)
    GUI.SetSize (canvas, canvasWidth, canvasHeight)
    GUI.SetSliderSize (h, canvasWidth + 1)
    GUI.SetPosition (v, 15 + canvasWidth,
        bottomBorder + GUI.GetScrollBarWidth - 1)
    GUI.SetSliderSize (v, canvasHeight + 1)
    GUI.SetSliderMinMax (h, 0, picWidth - 1)
    GUI.SetSliderMinMax (v, 0, picHeight - 1)
    GUI.SetScrollAmount (h, 3, 100, canvasWidth)
    GUI.SetScrollAmount (v, 3, 100, canvasHeight)
    GUI.SetSliderValue (h, 0)
    GUI.SetSliderValue (v, picHeight)
    GUI.Show (canvas)
    GUI.Show (h)
    GUI.Show (v)
    ScrollPic (0)
end LoadFile

procedure LoadFileButton
    var fileName : string := GUI.GetText (fileNameField)
    LoadFile (fileName)
end LoadFileButton

View.Set ("graphics:265;295")

% We place the canvas first and everything else is placed
% relative to the canvas.
canvas := GUI.CreateCanvas (leftBorder,
    bottomBorder + GUI.GetScrollBarWidth, maxSize, maxSize)
h := GUI.CreateHorizontalScrollBarFull (GUI.GetX (canvas),
    GUI.GetY (canvas) - GUI.GetScrollBarWidth,
    GUI.GetWidth (canvas), 0, 100, 0, ScrollPic, 3, 100, maxSize)
v := GUI.CreateVerticalScrollBarFull (
    GUI.GetX (canvas) + GUI.GetWidth (canvas),
    GUI.GetY (canvas), GUI.GetHeight (canvas), 0, 100,
    100, ScrollPic, 3, 100, maxSize)
fileNameField := GUI.CreateTextField (GUI.GetX (canvas),
    GUI.GetY (canvas) + GUI.GetHeight (canvas) + 10, 150, "",
    LoadFile)
loadButton := GUI.CreateButton (GUI.GetX (fileNameField) +
    GUI.GetWidth (fileNameField) + 20,
    GUI.GetY (fileNameField), 0, "Load File", LoadFileButton)
errorLabel := GUI.CreateLabel (GUI.GetX (canvas), 5, "")

% Set the initial picture and return if it is not found.
GUI.SetText (fileNameField, "Forest.bmp")
LoadFileButton
if pic = 0 then
    return
end if

```



```

loop
  exit when GUI.ProcessEvent
end loop

```

- Status Exported qualified.
 This means that you can only call the function by calling **GUI.SetScrollAmount**, not by calling **SetScrollAmount**.
- See also **GUI.CreateHorizontalScrollBar** and **GUI.CreateVerticalScrollBar**

GUI.SetSelection



- Syntax **GUI.SetSelection** (*widgetID*, *fromSel*, *toSel* : int)
- Description Sets the selected text in the text field specified by *widgetID*. The value of the *fromSel* and *toSel* parameters indicate the characters where the selection will begin and end. For example, if the text was "Hello there", setting *fromSel* to 2 and *toSel* to 5 would select "ell". Setting *fromSel* and *toSel* to -1 automatically selects the entire text.
- The *fromSel* parameter specifies the start of the selection. This ranges from 1 (before the first character) to the number of characters in the text + 1 (after the last character). A value of -1 for both *fromSel* and *toSel* selects the entire text.
- The *toSel* parameter specifies the end of the selection. This ranges from 1 (before the first character) to the number of characters in the text + 1 (after the last character). A value of -1 for both *fromSel* and *toSel* selects the entire text.
- Example The following program allows the user to type into a text field. When the user presses ENTER, it searches for any non-lowercase text and if it finds any, selects it to make it easy for the user to correct it. If all the input is lower-case text, the program terminates.

```

import GUI in "%oot/lib/GUI"

var textField, lbl : int

procedure CheckInput (s : string)
  for i : 1 .. length (s)
    if (s (i) < 'a' or 'z' < s (i)) and s (i) not= ' ' then
      GUI.SetSelection (textField, i, i + 1)
    return
  end if

```

```

        end for
        GUI.Quit
    end CheckInput

    textField := GUI.CreateTextField (100, 100, 200, "", CheckInput)
    lbl := GUI.CreateLabelFull (100 + GUI.GetWidth (textField) div 2,
        100 + GUI.GetHeight (textField),
        "Only Allows Lower Case Letters", 0, 0,
        GUI.CENTER + GUI.BOTTOM, 0)

loop
    exit when GUI.ProcessEvent
end loop

    GUI.SetLabel (lbl, "Program Finished!")

```

Status Exported qualified.

 This means that you can only call the function by calling **GUI.SetSelection**, not by calling **SetSelection**.

See also **GUI.CreateTextField**.

GUI.SetSize



Syntax **GUI.SetSize (*widgetID*, *width*, *height* : int)**

Description Changes the size of the widget specified by *widgetID*. If the widget is visible, its size is changed immediately, otherwise the widget will appear in its new size when the widget is next made visible. Note that the *width* and *height* parameters are not necessarily the actual width and height of the widget. For example, the *TextField* widget ignores the *height* parameter, calculating the widget's actual height from the height of the text in the *TextField*.

Example The following program resizes the button every time the button is pressed.

```

import GUI in "%oot/lib/GUI"

var button : int

procedure ResizeButton
    var newWidth, newHeight : int
    newWidth := Rand.Int (0, 200)
    newHeight := Rand.Int (0, 200)
    GUI.SetSize (button, newWidth, newHeight)
end ResizeButton

```

```

button := GUI.CreateButton (100, 100, 0, "Resize Button",
    ResizeButton)
loop
    exit when GUI.ProcessEvent
end loop

```

Status Exported qualified.
This means that you can only call the function by calling **GUI.SetSize**, not by calling **SetSize**.

GUI.SetSliderMinMax



Syntax **GUI.SetSliderMinMax** (*widgetID*, *min*, *max* : **int**)

Description Sets the minimum and maximum values of the slider or scroll bar specified by *widgetID*. The *min* parameter specifies the new minimum value of the slider or scroll bar. The *max* parameter specifies the new maximum value of the slider or scroll bar. The *max* parameter must be greater than the *min* parameter.
GUI.SetSliderMinMax redraws the thumb to take into account the new minimum and maximum. If the current value of the slider is outside the new minimum/maximum, then the value is adjusted appropriately.

Example See **GUI.SetScrollAmount** for an example of **GUI.SetSliderMinMax**.

Status Exported qualified.
This means that you can only call the function by calling **GUI.SetSliderMinMax**, not by calling **SetSliderMinMax**.

See also **GUI.CreateHorizontalScrollBar**, **GUI.CreateVerticalScrollBar**, **GUI.CreateHorizontalSlider**, and **GUI.CreateVerticalSlider**.

GUI.SetSliderReverse



Syntax	GUI.SetSliderReverse (<i>widgetID</i> : int)
Description	Sets a slider or scroll bar specified by <i>widgetID</i> into (or out of, if already into) "reverse mode". Normally, a slider or scroll bar is at its minimum value when the thumb is on the left hand side (bottom for a vertical slider). This reverses it, so the minimum value is when the thumb is at the right hand side (top for vertical sliders) of the track. Calling this routine a second time reverses it back to normal. This procedure redraws the slider to move the thumb to its new location.
Example	<p>The following program creates two sliders, one of which is reversed.</p> <pre>import GUI in "%oot/lib/GUI" View.Set ("graphics:300;70") var sBar, sBarLabel, reverseSBar, reverseSBarLabel : int procedure SBarMoved (value : int) GUI.SetLabel (sBarLabel, intstr (value)) end SBarMoved procedure ReverseSBarMoved (value : int) GUI.SetLabel (reverseSBarLabel, intstr (value)) end ReverseSBarMoved sBar := GUI.CreateHorizontalScrollBar (10, 10, 250, 50, 150, 50, SBarMoved) sBarLabel := GUI.CreateLabel (GUI.GetX (sBar) + GUI.GetWidth (sBar) + 10, 10, "50") reverseSBar := GUI.CreateHorizontalScrollBar (10, 40, 250, 50, 150, 50, ReverseSBarMoved) GUI.SetSliderReverse (reverseSBar) reverseSBarLabel := GUI.CreateLabel (GUI.GetX (reverseSBar) + GUI.GetWidth (reverseSBar) + 10, 40, "50") loop exit when GUI.ProcessEvent end loop</pre>
Status	Exported qualified. This means that you can only call the function by calling GUI.SetSliderReverse , not by calling SetSliderReverse .
See also	GUI.CreateHorizontalScrollBar , GUI.CreateVerticalScrollBar , GUI.CreateHorizontalSlider , and GUI.CreateVerticalSlider .

GUI.SetSliderSize



Syntax	GUI.SetSliderSize (<i>widgetID</i> , <i>length</i> : int)
Description	Changes the length of a slider or scroll bar specified by <i>widgetID</i> to the value specified by the <i>length</i> parameter. Redraws the slider or scroll bar and changes the position of the thumb to take into account the new size of the slider or scroll bar.
Example	See GUI.SetScrollAmount for an example of GUI.SetSliderSize .
Status	Exported qualified. This means that you can only call the function by calling GUI.SetSliderSize , not by calling SetSliderSize .
See also	GUI.CreateHorizontalScrollBar , GUI.CreateVerticalScrollBar , GUI.CreateHorizontalSlider , and GUI.CreateVerticalSlider .

GUI.SetSliderValue



Syntax	GUI.SetSliderValue (<i>widgetID</i> , <i>value</i> : int)
Description	Sets the value of a slider or scroll bar specified by <i>widgetID</i> to <i>value</i> . It moves the thumb on the slider or scroll bar to the appropriate location and calls the slider's <i>action procedure</i> with the new value.
Example	See GUI.SetScrollAmount for an example of GUI.SetSliderValue .
Status	Exported qualified. This means that you can only call the function by calling GUI.SetSliderValue , not by calling SetSliderValue .
See also	GUI.GetSliderValue for reading a slider or scroll bar's value. See also GUI.CreateHorizontalScrollBar , GUI.CreateVerticalScrollBar , GUI.CreateHorizontalSlider , and GUI.CreateVerticalSlider .

GUI.SetText



Syntax	GUI.SetText (<i>widgetID</i> : int , <i>text</i> : string)
Description	Sets the text of a text field specified by <i>widgetID</i> to <i>text</i> . The selection is set to 1, 1 (i.e. the cursor is at the beginning of the text).
Status	Exported qualified. This means that you can only call the function by calling GUI.SetText , not by calling SetText .
Example	<p>The following program converts all lower case input in the text field to upper case when the user presses ENTER.</p> <pre> import GUI in "%oot/lib/GUI" var textField, lbl : int procedure CheckInput (s : string) var newString : string := "" for i : 1 .. length (s) if 'a' <= s (i) and s (i) <= 'z' then newString += chr (ord (s (i)) - 32) else newString += s (i) end if end for GUI.SetText (textField, newString) GUI.SetSelection (textField, -1, -1) end CheckInput textField := GUI.CreateTextField (100, 100, 200, "", CheckInput) lbl := GUI.CreateLabelFull (100 + GUI.GetWidth (textField) div 2, 100 + GUI.GetHeight (textField), "Converts to Upper Case", 0, 0, GUI.CENTER + GUI.BOTTOM, 0) loop exit when GUI.ProcessEvent end loop </pre>
Status	Exported qualified. This means that you can only call the function by calling GUI.SetText , not by calling SetText .
See also	GUI.CreateTextField .

GUI.SetXOR



Syntax	GUI.SetXOR (<i>widgetID</i> : int , <i>xor</i> : boolean)
Description	Sets the "xor mode" of the canvas specified by <i>widgetID</i> . If the <i>xor</i> parameter is set to true , the canvas is set to <i>xor mode</i> . When in <i>xor mode</i> , all the <i>Draw...</i> procedures of a canvas are treated as if the View.Set ("xor") statement had been executed before the <i>Draw</i> procedure.
Example	See GUI.SetDisplayWhenCreated for an example of GUI.Show . <pre> import GUI in "%oot/lib/GUI" View.Set ("graphics:400;300") var canvas1, label1, canvas2, label2 : int canvas1 := GUI.CreateCanvas (10, 20, maxx div 2 - 20, maxy - 30) label1 := GUI.CreateLabelFull (10, 2, "XOR", maxx div 2 - 20, 0, GUI.CENTER, 0) canvas2 := GUI.CreateCanvas (maxx div 2 + 10, 20, maxx div 2 - 20, maxy - 30) label2 := GUI.CreateLabelFull (maxx div 2 + 10, 2, "Normal", maxx div 2 - 20, 0, GUI.CENTER, 0) GUI.SetXOR (canvas1, true) for i : 1 .. 20 var x : int := Rand.Int (0, maxx div 2 - 20) var y : int := Rand.Int (0, maxy - 20) var c : int := Rand.Int (1, 15) GUI.DrawFillStar (canvas1, x - 20, y - 20, x + 20, y + 20, c) end for GUI.SetXOR (canvas2, false) for i : 1 .. 20 var x : int := Rand.Int (0, maxx div 2 - 20) var y : int := Rand.Int (0, maxy - 20) var c : int := Rand.Int (1, 15) GUI.DrawFillStar (canvas2, x - 20, y - 20, x + 20, y + 20, c) end for </pre>
Status	Exported qualified. This means that you can only call the function by calling GUI.SetXOR , not by calling SetXOR .
See also	GUI.CreateCanvas .

GUI.Show



Syntax	GUI.Show (<i>widgetID</i> : int)
Description	<p>Shows a widget specified by <i>widgetID</i>. Used in conjunction with GUI.Hide to show and hide widgets. Hidden widgets cannot get events (i.e. respond to keystrokes or mouse clicks). If an active text field (see text field) is hidden, then any keystrokes in the window will be ignored.</p> <p>In most cases where a widget is to appear, then disappear, then appear again, it is advised to create the widget once and hide it until it is to appear, whereupon GUI.Show is called. When the user is finished with the widget, the widget is hidden using GUI.Hide. This saves the overhead of creating and disposing of the same widget several times.</p>
Example	See GUI.SetDisplayWhenCreated for an example of GUI.Show .
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.Show, not by calling Show.</p>
See also	GUI.Hide .

GUI.ShowMenuBar



Syntax	GUI.ShowMenuBar
Description	Shows the menu bar in the selected window.
Example	See GUI.SetMouseEventHandler for an example of GUI.HideMenuBar .
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling GUI.ShowMenuBar, not by calling ShowMenuBar.</p>
See also	GUI.HideMenuBar . See also GUI.CreateMenu .

handler exception handler

Dirty

Syntax	<p>A <i>exceptionHandler</i> is:</p> <pre>handler (<i>id</i>) <i>statementsAndDeclarations</i> end handler</pre>
Description	<p>An exception handler is an optional block of statements and declarations in a subprogram (or process). It is activated when the program (or process) fails. This occurs, for example when dividing by zero.</p>

Example	<p>This program parses the input stream using a stack. If the stack overflows (its top exceeds its maximum), a quit statement in the <i>push</i> procedure aborts the parsing and gives control to the exception handler in the <i>parse</i> procedure. The <i>parse</i> procedure calls <i>parseExpn</i> which calls <i>push</i>. If <i>push</i> overflows the stack, it executes a quit and control is passed to the exception handler in the <i>parse</i> procedure. The interrupted procedures (<i>parseExpn</i> and <i>push</i>) are terminated and their local variables are deleted.</p>
---------	---

```
const stackOverflow := 500  
const maxTop := 100  
var top : 0 .. maxTop := 0  
var stack : array 1 .. maxTop of int  
  
procedure push ( i : int )  
    if top = maxTop then  
        quit : stackOverflow  
    end if  
    top := top + 1  
    stack ( top ) := i  
end push  
...  
procedure parse  
    handler ( exceptionNumber )  
        put "Failure number ", exceptionNumber  
        case exceptionNumber of  
            label stackOverflow :  
                put "Stack has overflowed!!"  
            ... other exceptions handled here ...  
            label :  
                % Unexpected failures  
                quit >  
                % Pass exception further  
            end case  
        end handler  
    parseExn  
        % Eventually push is called  
end parse
```

Details

See the **quit** statement for an explanation of its *quitReason* (*stackOverflow* in the first **quit** statement above) and its *guiltyParty* (> in the second **quit** statement, meaning the exception is due to causes outside of this handler).

An exception handler can appear only in the body of a subprogram (or process), just preceding the declarations and statements. The form of a procedure is:

```

procedure [ pervasive ] id
  [ ( [ paramDeclaration { paramDeclaration } ] ) ]
  [ import [ [ var ] id { , [ var ] id } ] ]
  [ pre trueFalseExpn ]
  [ init id := expn { , id := expn } ]
  [ post trueFalseExpn ]
  [ exceptionHandler ]
  statementsAndDeclarations
end id

```

Exactly the same declarations and statements can appear in a handler as can appear in the subprogram body following the handler. In the absence of exceptions, handlers have no observable effect. A particular handler is activated (it becomes ready to handle an exception) when it is encountered during execution. It remains active until the subprogram (or process) containing it has completed, or the handler is given control. Activation of a handler when a previous handler is already active will cause exceptions to be passed to the newly-activated handler. In other words, handlers have a dynamic scope that begins when the exception handler is encountered and ends when the subprogram (or process) containing the handler has terminated or the handler is given control.

When a handler is given control, it becomes, in effect, a replacement for the declarations and statements following it. If the handler is in a function, it must terminate with a **result** statement or with a **quit**. If the handler is in a procedure (or process), the handler must terminate with a **return**, a **quit**, or by encountering the end of the handler (which is equivalent to a **return**).

When a handler terminates with a **result** or **return** statement (or by reaching the end of a procedure's handler), the subprogram's **post** condition (if any) must be true. A **quit** statement does not need to establish the **post** condition.

Programming with exception handlers easily leads to incomprehensible software, due to the difficulty of keeping track of the flow of control. One of the most insidious situations is when an exception occurs in a module, class or monitor and is propagated outside of the unit. This can leave the contained data in an inconsistent state; in the case of a monitor, it is left locked forever. To avoid this possibility, you can use a handler in each exported subprogram. If an exception in a process is not handled, the entire program is aborted. If an implementation allocates dynamic arrays on the heap, an exception may prevent the deallocation of such an array.

Without exception handling, a program executes according to the language definition or else is aborted. If an exception handler is active, instead of aborting, control is given to the handler. The *quitNumber* for a system-detected failure is implementation-dependent. There is a file "*%exceptions*" which lists these numbers. The user program can simulate a system exception by doing a **quit** with the corresponding number.

If the user turns off checking explicitly, the system may not detect failures. In some cases the failure may yield incorrect data or arbitrary behavior.

Some exceptions are unpredictable or implementation-dependent. For example, in $x := 24 \operatorname{div} i + 24 / i$, if i is zero, the exception could be either an integer or a real division by zero, because the order of evaluation is implementation-dependent.

hasch has character function

Syntax **hasch : boolean**

Description The **hasch** procedure is used to determine if there is a character that has been typed but not yet been read.

Example The *flush* procedure gets rid of any characters that have been typed but not yet read.

```
procedure flush
  var ch : string ( 1 )
  loop
    exit when not hasch
    getch ( ch )    % Discard this character
  end loop
end flush
```

Details The screen should be in a "screen" or "graphics" mode. See the **setscreen** procedure for details. If the screen is not in one of these modes, it will automatically be set to "screen" mode.

See also **getch** and **getchar**.
See also predefined unit **Input**.

id (identifier) name of an item in a program

Description	<p>Variables, constants, types, procedures, etc. in Turing programs are given names such as <i>incomeTax</i>, <i>x</i>, and <i>height</i>. These names are called identifiers (<i>ids</i>).</p> <p>An identifier must start with a letter (large or small) or an underscore (<code>_</code>) and can contain up to 50 characters, each of which must be a letter, a digit (0 to 9) or an underscore (<code>_</code>). Large and small letters are considered distinct, so that <i>A</i> and <i>a</i> are different names. This differs from Pascal where large and small letters in names are equivalent.</p> <p>Every character in a name is significant in distinguishing one name from another.</p> <p>By convention, words that make up an identifier are capitalized (except the first one), as in <i>incomeTax</i> and <i>justInTime</i>.</p> <p>An item in a Turing program cannot be given the same name as a keyword such as get or as a reserved word such as index. See Appendix A for a list of keywords and reserved words. As well, there are some identifiers that are used by the Turing error recovery procedures and are thus unavailable for use as identifiers. Specifically, they are: <i>endif</i>, <i>elseif</i>, <i>endloop</i> and <i>endfor</i>.</p>
-------------	--

if statement

Syntax	<p>An <i>ifStatement</i> is:</p> <pre>if trueFalseExpn then statementsAndDeclarations { elseif trueFalseExpn then statementsAndDeclarations } [else statementsAndDeclarations] end if</pre>
--------	---

Description	<p>An if statement is used to choose among a set of statements (and declarations). One set (at most) is chosen and executed and then execution continues just beyond end if.</p> <p>The expressions (the <i>trueFalseExpressions</i>) following the keyword if and each elsif are checked one after the other until one of them is found to be true, in which case the statements (and declarations) following the corresponding then are executed. If none of these expressions evaluates to true, the statements following else are executed. If no else is present and none of the expressions are true, no statements are executed and execution continues following the end if.</p>
Example	<p>Output a message based on value of mark.</p> <pre> if mark >= 50 then put "You pass" else put "You fail" end if </pre>
Example	<p>Output A, B, C, D or F depending on mark.</p> <pre> if mark >= 80 then put "A" elsif mark >= 70 then put "B" elsif mark >= 60 then put "C" elsif mark >= 50 then put "D" else put "F" end if </pre>
Example	<p>If x is negative, change its sign.</p> <pre> if $x < 0$ then $x := -x$ end if </pre>
Example	<p>If x is less than zero or greater than maxx, put a message.</p> <pre> if $x < 0$ or $x > \text{maxx}$ then put "Out of bounds!" end if </pre>
Example	<p>If the boolean <i>flag</i> is true and <i>name</i> is "stop", put a message and return.</p> <pre> if flag and name = "stop" then put "Exiting routine" return end if </pre>
Details	Several statements and declarations can appear after a particular then .
See also	case statements for another way to select among statements.

#if used for conditional compilation

Syntax	<p>A conditional compilation #if has the form:</p> <pre>#if <i>expn</i> then ... <i>any source text</i> ... { #elseif <i>expn</i> then ... <i>any source text</i> ... } [#else ... <i>any source text</i> ...] #end if</pre>
Description	<p>An #if construct supports compile time selection of sections of source text to make up a program (or unit of a program), in other words <i>conditional compilation</i>. Any arbitrary source text (characters) can be selected.</p> <p>Each of the selecting expressions (<i>expns</i>) have the form of a boolean expression, with the use of the operators and, or and not (but not =>) and parentheses. The short forms & and ~ are supported. The operands of the expressions must be <i>preprocessor flags</i>, which are set by a system-dependent mechanism not described here. A flag is considered to be true if it is explicitly set. If it is not explicitly set, it is considered false.</p> <p>Unlike other parts of the language, the #if, #elseif, #else and #end if constructs are not free format. Specifically, they must be placed by themselves on a single line.</p>
Example	<p>A pair of declarations is chosen if both <i>stats</i> and <i>debug</i> are set, otherwise the put statement is selected. The selected part becomes part of the program and the other parts are ignored.</p> <pre>#if <i>stats</i> and <i>debug</i> then var <i>count</i> : array 1 .. 5 of real var <i>message</i> : string #else put "Debugging message" #end if</pre>

implement clause

Syntax	An <i>implementClause</i> is: implement <i>implementItem</i>
Description	<p>An implement clause is used to specify that the module, monitor or class containing the clause is to be the implementation of another module, monitor or class. This implementation is a special kind of expansion. The module, monitor or class containing the clause gains access to (inherits) all the declarations inside the target item. See inherit clause for rules about expansions, which are also rules for implementations.</p> <p>The implement clause can only be used in a unit. See unit for the definition of a unit.</p>
Example	<p>Here is a <i>stack</i> module which defers all of its exported subprograms. This module is an interface but not an implementation. Following <i>stack</i> is the <i>stackBody</i> module that implements the <i>stack</i> module, giving the bodies for <i>stack</i>'s subprograms. Any call to <i>stack</i>'s <i>push</i> or <i>pop</i> procedures, such as <i>stack.push</i>("Ed"), will actually call the procedures given in <i>stackBody</i>.</p> <pre>module stack % Interface implement by stackBody % stackBody has implementation export push, pop deferred procedure push (s : string) deferred procedure pop (var s : string) end stack</pre> <p>Next comes the expansion which gives the bodies for the deferred procedures <i>push</i> and <i>pop</i>. The <i>stackBody</i> body also adds declarations for the <i>top</i> and <i>contents</i> variables.</p> <pre>module stackBody % Implementation implement stack % stack has interface var top : int := 0 var contents : array 1 .. 100 of string body procedure push % (s : string) top := top + 1 contents (top) := s end push body procedure pop % (var s : string) s := contents (top) top := top - 1 end pop end stackBody</pre>

Details

Module, monitor or class *D* can be in *C*'s **implement-by** clause if, and only if, *C* is in *D*'s **implement** clause. In other words, an interface must apply to exactly one implementation and vice versa. A module can **implement** only a module, a monitor only a monitor, and a class only a class. Classes (but not modules and monitors) can contain **inherit** clauses. A class cannot contain both an **inherit** and an **implement** clause.

An *implementItem* is one of:

- (a) *id*
- (b) *id in fileName*

The second form is used when the **implement** clause is for a separate **unit** and the imported item is in a file whose name is different from the item's name, as in:

```
implement ledger in "ledg.t"
```

The *fileName* must be an explicit character string, e.g., "ledg.t". See also **unit**. Parentheses are allowed around the items in **implement** clauses, as in:

```
implement ( ledger in "ledg.t" )
```

There is no restriction on the declarations that an interface may contain. In particular, an interface (any module, monitor or class containing an **implement-by** clause), can contain subprogram bodies and variable declarations, exactly as is the case in expansions. This is different from languages such as C++ in which there are strict rules limiting what you can put in an interface.

Even though *D* contains an **implement** clause, *D* can also contain an **implement-by** clause, which implies further implementation by further automatic expansion.

Suppose class *D* is in class *C*'s **implement-by** clause and that *p* is a pointer to class *C*:

```
var p : ^ C
```

Even though *C* is implemented by *D*, *p* remains a pointer to class *C*. Each creation of an object of class *C* actually creates an object of type *D*, for example:

```
new p          % Creates object of class D
```

Class *D*, which implements *C*, could also have an **implement-by** clause, which causes its implementation to be automatically created and so on. If another class *E* inherits *C*, this expansion does not include *D*.

If the **new** statement contains an explicit class name *E* that is a descendant of *C* (but not actually *C*), as in

```
new E, p
```

the object of the explicit class is created. If *E* has an **implement-by** clause, the expansion is created.

See also

unit, **module**, **monitor** and **class**. See also **implement by** clause, **inherit** clause, **export** list, and **import** list. See also **deferred** subprograms.

implement by clause

Syntax	An <i>implementByClause</i> is: implement by <i>implementByItem</i>
Description	<p>An implement-by clause is used to specify that a module, monitor or class <i>C</i> is to be automatically implemented by the <i>implementByItem</i>. <i>C</i> is called the <i>interface</i> and the <i>implementByItem</i>, which must contain an implement clause, is called the <i>implementation</i>. See implement clause for details and an example.</p> <p>The implement-by clause can only be used in a unit. See unit for the definition of a unit.</p> <p>An <i>implementByItem</i> is one of:</p> <ul style="list-style-type: none">(a) <i>id</i>(b) <i>id in fileName</i> <p>The second form is used when the implement-by clause is for a separate unit and the imported item is in a file whose name is different from the item's name, as in:</p> <p style="text-align: center;">implement by ledgerBody in "ledgbod.t"</p> <p>The fileName must be an explicit character string, e.g., "ledgbod.t". See also unit. Parentheses are allowed around the items in an implement-by clauses, as in:</p> <p style="text-align: center;">implement by (ledgerBody in "ledgbod.t")</p>

import list

Syntax	An <i>importList</i> is: import [<i>howImport</i>] <i>importItem</i> {, [<i>howImport</i>] <i>importItem</i> }
--------	---

Description An **import** list is used to specify those items that a procedure, function, module, monitor, or a class uses from outside of itself. Note that a function or procedure is not allowed to have an import list and thus automatically imports whichever functions or procedures are used by the function or procedure. The compiler determines the list automatically by looking to see what items are actually used.

Example In this example, the type *T* is imported into the *stack* **module** and used as the type that can be pushed onto or popped off the stack. Since no other items are imported, the only identifiers from outside of *stack* that can be used in it must be predefined, such as **sqrt**, or declared to be **pervasive**.

```
type T : string
...
module stack
  import T
  export push, pop
  var top : int := 0
  var contents : array 1..100 of T
  procedure push ... end push
  procedure pop ... end pop
end stack
```

Details The *importItem* is one of:

- (a) *id*
- (b) *id in fileName*

The second form is used in OOT when the list is the import list for a separate **unit** (or the main program), and the imported item is in a file whose name is different from the item's name, for example:

```
import ledger in "newledg.t"
```

The *fileName* must be an explicit character string. See also **unit**.

Parentheses are allowed around the items in an import lists, as in:

```
import ( ledger in "newledg.t" )
```

There are various ways to import items, as determined by *howImport*. The form of *howImport* is one of:

- (a) **var**
- (b) **const**
- (c) **forward**

Commonly the *howImport* is omitted, which means the default access for the item is the same access as the item has. In other words, a read-write item that is imported without a *howImport* is imported read-write. A read-only symbol that is imported without a *howImport* is imported read-only.

If the *importItem* is **forward**, the import list is part of a **forward** procedure or function declaration and the imported item is itself necessarily a procedure or function. See **forward** declarations for details and an example.

If the **import** list of a **module**, **monitor** or **class** is omitted, the implementation assumes that the list is **import()**, meaning that no items are imported. For example, a **module** must explicitly import any global identifiers that are not predefined or **pervasive**.

Circular (recursive) imports are not allowed. For example, if unit *A* imports *B* then *B* cannot import *A*. However, circular usage of separately compiled units is possible by separating the units into interfaces and bodies and having the bodies import the interfaces. For example, if *C* is the parent class of *D*, *D* can import *C*, but not vice versa.

In an expansion (or implementation), the import list of the expansion augments the import list of the parent.

An overriding subprogram (in an expansion) ignores the import list of the target subprogram and uses its own import list.

Turing initializes modules and monitors in order of importation.

Initialization begins with the main program, which first initializes its imports in the order given in its **import** list, and then initializes itself.

See also

unit, **module**, **monitor** and **class**. See also **export** list, **inherit** clause, **implement** clause and **implement by** clause.

in member of a set

Syntax

in

Description

The **in** operator determines if an element is in a set.

Example

```
type rankSet : set of 0 .. 10
var rankings : rankSet := rankSet ( 0 ) % The set {0}
...
      if 5 in rankings then ...           % Is 5 in the rankings set?
```

Description

The **not in** operator is exactly the opposite of **in**. For example, *7 not in rankings* means the same as **not** (*7 in rankings*).

The element is required to be in the set's index type. In the above example this is satisfied because element 5 is in the index type 0 .. 10.

The keyword **in** is also used in lists such as **import** lists. See **import** list.

See also

the **set** type, *infix operators*, and *precedence* of operators.

include source files

Syntax *An includeConstruct is:*

include *fileName*

Description An **include** is used to copy parts of files so that they become part of the Turing program. This copying is temporary, that is, no files are changed. The file name must be an explicit string constant such as "stdstuff".

Example On IBM PC compatible computers, there are arrow keys that produce character values such as 200 and 208. Let us suppose that a file called *arrows* contains definitions of these values:

```
const upArrow := 200
const downArrow := 208
const rightArrow := 205
const leftArrow := 203
```

These definitions can be included in any program in the following manner:

```
include "arrows"
...
var ch : string ( 1 )
getch ( ch )                                % Read one character
case ord ( ch ) of
  label upArrow :
    ...handle up arrow...
  label downArrow :
    ...handle down arrow...
  label rightArrow :
    ...handle right arrow...
  label leftArrow :
    ...handle left arrow...
  label :
    ...handle any other key...
end case
```

Details An include file can itself contain **include** constructs. This can continue to any level, although a circular pattern of includes would be a mistake, as it would lead to an infinitely long program.

It is common to save procedures, functions and modules in separate files. The files are collected together using **include**.

Details If the filename in the **include** statement starts with a "%", then Turing searches the system directory for the file. See the editor reference for the environment to see how to set the system directory. This method can be used to allow the system administrator to easily supply a set of routines in a file to a large number of users by placing it in one easy-to-find location.

Example	<p>If the system directory is set to "C:\TURING", then the line</p> <pre>include "%sorting.t"</pre> <p>will include the file "C:\TURING\SORTING.T" in the program.</p>
Details	<p>Under OOT, there are several system directories available. The "%oot" directory is the directory where all the OOT system files are located. The "%home" directory is the user's home directory.</p>
Example	<p>If the oot directory is set to "/usr/local/lib/oot" then the line</p> <pre>include "%oot/teacher/sorting.t"</pre> <p>will include the file "/usr/local/lib/oot/teacher/sorting.t" in the program.</p>

index find pattern in string function

Syntax	index (<i>s</i> , <i>patt</i> : string) : int
Description	The index function is used to find the position of <i>patt</i> within string <i>s</i> . For example, index ("chair", "air") is 3.
Example	<p>This program outputs 2, because "ill" is a substring of "willing", starting at the second character of "willing".</p> <pre>var word : string := "willing" put index (word, "ill")</pre>
Details	<p>If the pattern (<i>patt</i>) does not appear in the string (<i>s</i>), index returns 0 (zero). For example, here is an if statement that checks to see if string <i>s</i> contains a blank:</p> <pre>if index (<i>s</i>, " ") not= 0 then ...</pre> <p>The index is sometimes used to efficiently determine if a character is one of a given set of characters. For example, here is an if statement that checks to see if <i>ch</i>, which is declared using var <i>ch</i> : string (1), is a digit:</p> <pre>if index ("0123456789", <i>ch</i>) not= 0 then ...</pre> <p>If a string contains more than one occurrence of the pattern, the leftmost location is returned. For example, index ("pingpong", "ng") returns 3.</p> <p>If <i>patt</i> is the null string, the result is 1.</p>

indexType

Syntax	An <i>indexType</i> is one of: <ul style="list-style-type: none">(a) <i>subrangeType</i>(b) <i>enumeratedType</i>(c) <i>namedType</i> % Which is a subrange or enumerated type(d) char(e) boolean
Description	An index type defines a range of values that can be used as an array subscript, as a case selector, as a selector (tag) for a union type, or as the base type of a set type.
Example	<pre>var z : array 1 .. 9 of real % 0..9 is an index type type smallSet : set of 0 .. 2 % 0..2 is an index type</pre>

indirection operator (@)

Dangerous

Syntax	<i>targetType</i> @ (<i>expn</i>)
Description	The indirection operator @ is used to access values that lie at absolute machine addresses in the computer's memory. This is dangerous and implementation-dependent and can cause arbitrary corruption of data and programs.
Example	Copy the byte value at memory location 246 into <i>b</i> and then set that memory byte to zero. <pre>var b : nat1 % One byte natural number b := nat1 @ (246) nat1 @ (246) := 0</pre>
Details	The form of <i>targetType</i> must be one of: <ul style="list-style-type: none">(a) [<i>id</i> .] <i>typeId</i>(b) int, int1, int2 or int4

- (c) **nat**, **nat1**, **nat2** or **nat4**
- (d) **boolean**
- (e) **char** [(*numberOfCharacters*)]
- (f) **string** [(*maxLength*)]
- (g) **addressint**

In form (a) the beginning identifier *id* must be the name of a module, monitor or class that exports the *typeId*. Each of *numberOfCharacters* and *maxLength* must be compile time integer expressions. These are the same target types as in type cheats.

The indirection operator @ takes an integer as an address. This value must fit in the range of **addressint**. See **addressint**. See also **pointer** types and the ^ operator (which accesses objects located by pointers).

See also

cheat. See also *explicitIntegerConstant* (which explains how to write hexadecimal constants, which are often used for addresses).

infix operator

Syntax

An *infixOperator* is one of:

- (a) **+** % Integer and real addition; set union;
% string catenation
- (b) **-** % Integer and real subtraction; set difference
- (c) ***** % Integer and real multiplication; set intersection
- (d) **/** % Real division
- (e) **div** % Truncating integer division
- (f) **mod** % Modulo
- (g) **rem** % Remainder
- (h) ****** % Integer and real exponentiation
- (i) **<** % Less than
- (j) **>** % Greater than
- (k) **=** % Equal
- (l) **<=** % Less than or equal; subset
- (m) **>=** % Greater than or equal; superset
- (n) **not=** % Not equal
- (o) **and** % And (boolean conjunction)

(p)	or	% Or (boolean disjunction)
(q)	=>	% Boolean implication
(r)	in	% Member of set
(s)	not in	% Not member of set
(t)	shr	% Shift right
(u)	shl	% Shift left
(v)	xor	% Exclusive OR

Description An *infix operator* is placed between two values or *operands* to produce a third value. For example, the result of 5 + 7 is 12. In some cases the meaning of the operator is determined by its operands. For example, in "pine" + "apple", the + operator means string catenation while in 5 + 7 it means integer addition. There are also *prefix operators* (**-**, **+** and **not**), which are placed in front of a single value. See *prefix operator*.

In expressions with several operators, such as 3 + 4 * 5, the *precedence* rules determine the order in which the operation is done (see *precedence* for a listing of these rules). In this example, the multiplication is done before the addition, so the expression is equivalent to 3 + (4 * 5).

The numerical (integer or real) operators are **+**, **-**, *****, **/**, **div**, **mod**, and ******. All of these except **div** produce a **real** result when at least one of their operands is **real**. If both operands are integers, the result is an integer except in the case of **real** division (**/**) which always produces a **real** result regardless of the operands.

The **div** operator is like **real** division (**/**), except that it always produces an integer result, truncating any fraction to produce the nearest integer in the direction of zero.

The **mod** operator is the *modulo* and the **rem** operator is the *remainder*. The sign of the result of **mod** operator is the same as the sign of the second operand. The **rem** operator operates like the **mod** operator in Turing (and in most other languages). It produces the remainder, which is the difference between **real** division (**/**) and integer division (**div**). When both operands are positive, this is the *modulo*. For example, 14 **mod** 10 is 4. If one of the operands is negative, a negative answer may result, for example, -7 **mod** 2 is -1. See also the **int** and **real** types.

The comparison operators (**<**, **>**, **=**, **<=**, **>=**, **not=**) can be applied to numbers as well as to enumerated types. They can also be applied to strings to determine the *ordering* between strings (see **string** type for details). Arrays, records, unions and collections cannot be compared. Boolean values (**true** and **false**) can be compared only for equality (**=** and **not=**); the same applies to **pointer** values. Set values can be compared using **<=** and **>=**, which are the subset and superset operators. The **not=** operator can be written as **~=**.

Strings are manipulated using catenation (**+**) as well as substring expressions (see *substring*) and the **index** function (see **index**). See also the **string** type.

The operators to combine true/false values are **and**, **or**, and **=>** (implication), as well as equality (**=** and **not=**). See also the **boolean** type.

The set operators are union (**+**), intersection (*****), set difference (**-**), subset (**<=**), superset (**>=**), and membership (**in** and **not in**). See also the **set** type.

The **shr** (shift right), **shl** (shift left) and **xor** (exclusive OR) operators accept and produce natural numbers. See **shr**, **shl**, and **xor**.

inherit inheritance clause

Syntax	An <i>inheritClause</i> is: inherit <i>inheritItem</i>
Description	An inherit clause specifies that the class containing the clause is to be an expansion of another class. This expansion is called <i>inheritance</i> . The class containing the clause gains access to (inherits) all the declarations inside the target item. Expansions are used to add new declarations and exports and to support <i>polymorphism</i> (overriding subprograms).
Example	Here is an example of a stack class. Following it, we show another class, called <i>stackWithDepth</i> , that inherits <i>stack</i> by adding a function called <i>depth</i> .

```

class stack
  export push, pop

  var top : int := 0
  var contents : array 1 .. 100 of string

  procedure push ( s : string )
    top := top + 1
    contents (top) := s
  end push

  procedure pop ( var s : string )
    s := contents ( top )
    top := top - 1
  end pop
end stack

```

Next comes an expansion, which inherits the internal declarations of the stack class and adds the *depth* function.

```

class stackWithDepth
  inherit stack
  export depth
  function depth : int
    result top

```

end *push*

end *stackWithDepth*

Details

Objects of the inherited class *stackWithDepth* are like objects of the parent class *stack*, except there is an additional exported function named *depth*.

An *inheritItem* is one of:

(a) *id*

(b) *id in fileName*

The second form is used when the *inherit* clause is for a separate **unit** and the imported item is in a file whose name is different from the item's name, for example:

inherit *ledger in* "newledg.t"

The *fileName* must be an explicit character string, e.g., "newledg.t". Parentheses are allowed around the item in an *inherit* clause, as in:

inherit (*ledger in* "newledg.t")

There is a special form of **inherit** clause, called an **implement clause**, that is used to separate an interface from an implementation. Modules and monitors, as well as classes, use these clauses. See **implement** clause and **implement by** clause.

If class *D* inherits class *C*, we say that *C* is the *parent* and *D* is the *child*.

Class *B* is said to be an *ancestor* of class *D* (and *D* is the *descendant* of *B*) if *B* and *D* are the same class, or if *B* is the parent of *D*, or if *B* is the parent of the parent of *D*, etc. We write this as follows:

$B \leq D$ % *B* is an ancestor of *D*

If *B* is an ancestor of *D* but not the same as *D*, we say *B* is a *strict* ancestor of *D*. We write this as:

$B < D$ % *B* is a strict ancestor of *D*

We also use the notations $D \geq B$, $D > B$ and $D = B$ with the obvious meanings. All of these notations can be used in a program. Their main use is in conjunction with **objectclass**, which determines the class of an object located by a pointer. For example, if *p* is declared to be a pointer to a *stack*, we can write the following to see if *p* currently locates an object with the *depth* operation:

% Does the object located by *p* have the *depth* operation

if *stackWithDepth* \leq **objectclass**(*p*) **then**

A pointer that locates an object created as class *E* can be assigned to a pointer to class *B*, only if *B* is an ancestor of *E*. For example, a pointer to an object that is a *stackWithDepth* can be assigned to a pointer to *stack*, but not vice versa. The pointer **nil** can be assigned to any pointer variable, but the value **nil**(*C*) can only be assigned to a pointer to an ancestor of *C*.

An object (located by a pointer) can be assigned to another object only if they were created as objects of the same class. However, assignment of objects that are monitors or that contain dynamic arrays or collections is not allowed.

Circular (recursive) inherits are not allowed. For example, if unit *B* inherits *A* then *A* cannot inherit *B*. Only one item is allowed in an inherit clause; in other words, Turing supports *single* inheritance but not *multiple* inheritance.

See **implement** clause for a special kind of expansion that separates a module, monitor or class' interface from its implementation. See **class** for an example of polymorphism, in which an inheriting class overrides subprograms of its parent class.

The initialization of a module, a monitor or an object is immediately preceded by the initialization of the item that it inherits or implements (if any). Correspondingly, if the item has an **implement by** clause, the implementation is initialized immediately after the initialization of the current item.

Within a class *C*, with ancestor *B*, you can force a call to exported subprogram *p* using the form *C.p* (or *B.p*). This calls the subprogram declared in *C* (or in *B* in the case of *B.p*), regardless of the actual class of the object and any overriding of *p*. This is similar to the notation *C::p* of the C++ language. This notation can only be used inside class *C*.

See also **unit**, **module**, **monitor** and **class**. See also **export** list, **import** list, **implement** clause, **implement by** clause and **deferred** subprogram. See also **objectclass**.

init array initialization

Syntax **init**

Description The **init** (initialization) keyword is used for two different purposes in Turing. The most common is for initializing arrays, records and unions. The less common is for recording parameter values in subprograms for later use in **post** conditions.

Example

```
var mensNames : array 1 .. 3 of string :=
    init ( "Tom", "Dick", "Harry" )
put mensNames ( 2 )                     % This outputs Dick
var names : array 1 .. 2, 1 .. 3 of string :=
    init ( "Tom", "Dick", "Harry",
          "Alice", "Barbara", "Cathy" )
    put names ( 2, 1 )   % This outputs Alice
```

Details	The order of initializing values for multi-dimensional arrays is based on varying the right subscripts (indexes) most rapidly. This is called <i>row major order</i> . Initialization of records and unions is analogous to initializing arrays. Values are listed in the order in which they appear in the type. See array , record , and union types.
Example	<p>This procedure is supposed to set integer variable <i>i</i> to an integer approximation of its square root. The init clause records the initial value of <i>i</i> as <i>j</i> so it can be used in the post condition to make sure that the approximation is sufficiently accurate. The name <i>j</i> can be used only in the post condition and nowhere else in the procedure.</p> <pre> procedure <i>intSqrt</i> (var <i>i</i> : int) pre <i>i</i> >= 0 init <i>j</i> := <i>i</i> post <i>abs</i> (<i>i</i> - <i>sqrt</i> (<i>j</i>)) <= 1 ... <i>statements to approximate square root</i> ... end intSqrt </pre>
See also	pre and post assertions and procedure and process declarations.

Input All

Description	<p>This unit contains the predefined procedures that deal with handling input on a character-by-character basis.</p> <p>All routines in the Input module are exported unqualified. (This means you can call the entry points directly.)</p>	
Entry Points	getch	Gets the next character in the keyboard buffer (procedure with a string (1) argument).
	hasch	Returns true if there are characters waiting in the keyboard buffer.
	getchar	Gets the next character in the keyboard buffer (function returning a char).
	Pause	Waits for a key to be pressed.

Input.getch

All

Syntax	getch (var <i>ch</i> : string (1))
Description	The getch procedure is used to input a single character without waiting for the end of a line. The parameter <i>ch</i> is set to the next character in the keyboard buffer (the oldest not-yet-read character).
Example	<p>This program contains a procedure called <i>getKey</i> which causes the program to wait until a key is pressed.</p> <pre>View.Set ("graphics") procedure <i>getKey</i> var <i>ch</i> : string (1) getch (<i>ch</i>) end <i>getKey</i> for <i>i</i> : 1 .. 1000 put <i>i</i> : 4, " Pause till a key is pressed" <i>getKey</i> end for</pre>
Details	<p>The screen should be in a "screen" or "graphics" mode. See the View.Set procedure for details. If the screen is not in one of these modes, it will automatically be set to "screen" mode.</p> <p>Some keys, such as the left arrow key, insert key, delete key, and function keys do not produce ordinary character values. These keystrokes are returned by getch as special values. See Appendix D for these values.</p>
Status	<p>Exported unqualified.</p> <p>This means that you can call the function by calling getch or by calling Input.getch.</p>
See also	hasch (has character) procedure which is used to see if a character has been typed but not yet read.

Input.getchar



Syntax	getchar : char
Description	The getchar function is used to input a single character without waiting for the end of a line. The next character in the keyboard buffer (the oldest not-yet-read character) is returned.
Example	<p>This program contains a procedure called <i>getKey</i> which causes the program to wait until a key is pressed.</p> <pre>View.Set ("graphics") procedure <i>getKey</i> var <i>ch</i> : char <i>ch</i> := getchar end <i>getKey</i> for <i>i</i> : 1 .. 1000 put <i>i</i> : 4, " Pause till a key is pressed" <i>getKey</i> end for</pre>
Details	<p>The screen should be in a "screen" or "graphics" mode. See the View.Set procedure for details. If the screen is not in one of these modes, it will automatically be set to "screen" mode.</p> <p>Some keys, such as the left arrow key, insert key, delete key, and function keys do not produce ordinary character values. These keystrokes are returned by getch as special values. See Appendix D for these values.</p>
Status	<p>Exported unqualified.</p> <p>This means that you can call the function by calling getchar or by calling Input.getchar.</p>
See also	hasch (has character) procedure which is used to see if a character has been typed but not yet read.

Input.hasch

All

Syntax	hasch : boolean
Description	The hasch procedure is used to determine if there is a character that has been typed but not yet been read.
Example	<p>The <i>flush</i> procedure gets rid of any characters that have been typed but not yet read.</p> <pre>procedure flush var ch : string (1) loop exit when not hasch getch (ch) % Discard this character end loop end flush</pre>
Details	The screen should be in a "screen" or "graphics" mode. See the View.Set procedure for details. If the screen is not in one of these modes, it will automatically be set to "screen" mode.
Status	Exported unqualified. This means that you can call the function by calling hasch or by calling Input.hasch .
See also	getch and getchar .

Input.Pause

All

Syntax	Input.Pause
Description	<p>The Input.Pause procedure simply waits for a key to be pressed and then returns. It echoes the key pressed if echo mode is set. (See View.Set for setting echo mode)</p> <p>This subprogram helps avoid having to declare a variable declaration and then make a call to getch or getchar.</p>

Example This program pauses after every name read from the file.

```
var f: int
open : f, "data", get
loop
  exit when eof (f)
  get : f, name : *
  put name
  Input.Pause
end loop
```

Status Exported qualified.

This means that you can only call the function by calling **Input.Pause**, not by calling **Pause**.

int integer type

Syntax **int**

Description The **int** (integer) type has the values ... -2, -1, 0, 1, 2 ... Integers can be combined by various operators such as addition (+) and multiplication (*). Integers can also be combined with **real** numbers, in which case the result is generally a real number. An integer can always be assigned to a real variable, with implicit conversion to **real**.

Example

```
var counter, i : int
var j : int := 9
var tax := 0            % The type is implicitly int because 0 is an integer
```

Details See also *explicitIntegerConstant*. The **real** type is used instead of **int** when values have fractional parts as in 16.837. See the **real** type for details.

The operators on integers are +, -, * (multiply), **div** (truncating integer division), **mod** (integer remainder), ** (exponentiation), as well as comparisons (+, **not=**, >, >=, <, <=). The operators **and**, **or** and **xor** can be applied to non-negative integer values. The bit-wise boolean result is produced as an integer (actually, as a natural number). The **shr** (shift right) and **shl** (shift left) operators are also introduced.

Real numbers can be converted to integers using **ceil** (ceiling), **floor**, and **round** (see descriptions of these functions). Integers can be converted to real numbers using **intreal**, but in practice this is rarely used, because an integer value used in place of a real value will be automatically converted to real.

Integers can be converted to strings and back using **intstr** and **strint**. Integers can be converted to corresponding ASCII (or EBCDIC) characters using **chr** and **ord**. See the descriptions of these functions.

Pseudo-random sequences of integers can be generated using **randint**. See **randint**.

In current implementations of Turing, the range of integers is from -2147483647 to 2147483647. In other words, the maximum size of integer is $2^{31} - 1$. See **maxint**. This range exists because integers are stored in 4 bytes. The remaining negative value, -2147483648 records uninitialization. The types **int1**, **int2** and **int4** specify integers that fit into 1, 2 or 4 bytes. The **int_n** types (**int1**, **int2** and **int4**) are not checked for initialization and allow all their bit patterns as numbers.

The natural number type **nat** allows only the non-negative values: 0, 1, 2, 3, ... Natural number values can be used whenever integer values are expected and vice versa, given that the value does not exceed the range of the expected type.

See also **nat** and **int_n**.

int_n n-byte integer type

Dirty

Syntax

- (a) **int1** % 1-byte integer
- (b) **int2** % 2-byte integer
- (c) **int4** % 4-byte integer

Description The **int_n** (*n*-byte integer) types are machine-dependent types that occupy a specified number of bytes. By contrast, the **int** type is in principle a machine-independent and mathematical type (it overflows, however, when the value is too large or small, that is, when the value does not fit into 4 bytes).

Example

```
var counter1 : int1 % Range is -128 .. 127
var counter2 : int2 % Range is -32768 .. 32767
var counter4 : int4 % Range is -2147483648 .. 2147483647
```

Details In current implementations of Turing, the range of the **int** is -2147483647 to 2147483647, which means that the **int4** type allows one more value, -2147483648. This extra value is used in **int** to represent the state of being initialized. The **int_n** types allow use of all possible values that fit into *n* bytes and thereby cannot check for initialization.

The **int n** types are like the C language types *short int*, *int*, and *long int*, except that the number of bytes occupied by the C types depends on the particular C compiler.

See also the **nat n** types which are n byte natural (non-negative) values. See also **int** and **nat**.

intreal integer-to-real function

Syntax **intreal ($i : \text{int}$) : real**

Description The **intreal** function is used to convert an integer to a **real** number. This function is rarely used, because in Turing, an **integer** value can be used where ever a **real** value is required. When the **integer** value is used where a **real** value is required, the **intreal** function is implicitly called to do the conversion from **int** to **real**.

See also **floor**, **ceil** and **round** functions.

intstr integer-to-string function

Syntax **intstr ($i : \text{int}$ [, $width : \text{int}$ [, $base : \text{int}$]]) : string**

Description The **intstr** function is used to convert an integer to a string. The string is equivalent to i , padded on the left with blanks as necessary to a length of $width$, written in the given number $base$. For example, **intstr** (14, 4, 10) = "bb14" where b represents a blank. The $width$ and $base$ parameters are both optional. If they are omitted, the string is made just long enough to hold the value, and the number base is 10. For example, **intstr** (14, 4) = "bb14" and **intstr** (-23) = "-23".

The $width$ parameter must be non-negative. If $width$ is not large enough to represent the value of i , the length is automatically increased as needed.

The string returned by **intstr** is of the form:

{blank}[-]digit{digits}

where {blank} means zero or more blanks, [-] means an optional minus sign, and digit{digit} means one or more digits. The leftmost digit is either non-zero or else a single zero digit. In other words, leading zeros are suppressed.

The letters A, B, C ... are used to represent the digit values 10, 11, 12, ... The *base* must be in the range 2 to 36 (36 because there are ten digits and 26 letters). For example, **intstr** (255, 0, 16) = "FF".

The **intstr** function is the inverse of **strint**, so for any integer *i*,

strint (**intstr** (*i*)) = *i*.

See also **chr**, **ord** and **strint** functions. See also the **natstr** and **strnat** functions. See also *explicitIntegerConstants* for the way to write non base 10 values in a program.

invariant assertion

Syntax An *invariantAssertion* is:

invariant *trueFalseExpn*

Description An **invariant** assertion is a special form of an **assert** statement that is used only in **loop** and **for** statements and in modules, monitors, and classes. It is used to make sure that a specific requirement is met. This requirement is given by the *trueFalseExpn*. The *trueFalseExpn* is evaluated. If it is true, all is well and execution continues. If it is false, execution is terminated with an appropriate message. See **assert**, **loop** and **for** statements and the **module** declarations for more details.

Example This program uses an invariant in a **for** loop. The invariant uses the function *nameInList* to specify that a key has not yet been found in an array of names.

```
var name : array 1 .. 100 of string
var key : string
... input name and key ...

function nameInList ( n : int ) : boolean
  for i : 1 .. n
    if key = name ( i ) then
      result true
    end if
  end for
  result false
end nameInList
```

```

for j : 1 .. 100
  invariant not nameInList ( j - 1)
  if key = name ( j ) then
    put "Found name at ", j
    exit
  end if
end loop

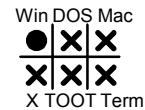
```

Joystick



Description	<p>This unit contains the predefined subprogram that deal with reading the joystick in a Turing program. The routines allow you to get the current joystick position and whether either one of the two buttons are pressed.</p> <p>All routines in the Joystick module are exported qualified (and thus must be prefaced with "Joystick."). All the constants are exported unqualified and thus do not need the Joystick prefix.</p>
Entry Points	<p>joystick1, joystick2 joystick name constants (exported unqualified)</p> <p>GetInfo Reads the current value of a joystick and status of the joystick buttons.</p>

Joystick.GetInfo



Syntax	<p>Joystick.GetInfo (<i>joystick</i> : int, var <i>xPos</i>, <i>yPos</i> : int, <i>btn1Pressed</i>, <i>btn2Pressed</i> : boolean)</p>
Description	<p>Reads the position and button status of the joystick specified by the <i>joystick</i> parameter. The <i>x</i> and <i>y</i> parameter are returned in the <i>xPos</i> and <i>yPos</i> parameters. If button 1 or button 2 on the joystick are currently pressed, <i>btn1Pressed</i> and <i>btn2Pressed</i> will be set to true. The <i>joystick</i> parameter can be either joystick1 or joystick2.</p> <p>The <i>x</i> and <i>y</i> positions vary from joyMin to joyMax. To use them with respect to a screen, the coordinates returned from Joystick.GetInfo must be translated into screen coordinates. The following formula can be used:</p> $screenX = \text{round} (maxx * (xPos - joyMin) / (joyMax - joyMin))$

$$screenY = \text{round} (maxy * (yPos - joyMin) / (joyMax - joyMin))$$

Details The **Joystick** module contains undocumented subprograms for those who need to access more than two buttons or axes on a joystick. Contact Holt Software if you need more information.

Example The following program outputs the current location of joystick #1 and draws a cursor on the screen to point out where it is showing.

```
var jx, jy, x, y, ox, oy : int := -1
var b1, b2, oB1, oB2 : boolean := false
loop
  Joystick.GetInfo (joystick1, jx, jy, b1, b2)
  % Convert joystick coordinates into screen coordinates.
  x = round (maxx * (jx - joyMin) / (joyMax - joyMin))
  y = round (maxy * (jy - joyMin) / (joyMax - joyMin))
  if x not= ox or y not= oy or b1 not= oB1 or b2 not= oB2 then
    Text.Locate (1, 1)
    put "x = ", x, " y = ", y, " b1 = ", b1, " b2 = ", b2
    View.Set ("xor")
    Draw.Line (ox - 10, oy, ox + 10, oy, brightred)
    Draw.Line (ox, oy - 10, ox, oy + 10, brightred)
    Draw.Line (x - 10, y, x + 10, y, brightred)
    Draw.Line (x, y - 10, x, y + 10, brightred)
    ox := x
    oy := y
    oB1 := b1
    oB2 := b2
  end if
end loop
```

Status Exported qualified.

This means that you can only call the function by calling **Joystick.GetInfo**, not by calling **GetInfo**.

length of a string function

Syntax **length (s : string) : int**

Description The **length** function returns the number of characters in the string. The string must be initialized. For example, **length**("table") is 5.

Example This program inputs three words and outputs their lengths.

```
var word : string
for i : 1 .. 3
```

```

get word
put length ( word )
end for

```

If the words are "cat", "robin" and "crow", the program will output 3, 5 and 4.

Details The **length** function gives the current length of the string. To find the maximum length of a string, use **upper**. For example, given the declaration **var s : string (10)**, **upper (s)** returns 10.

See also **upper**.

Limits All

Description This unit contains constants and functions used in determining the mathematical accuracy of the language.

All routines in the Limits module are exported qualified (and thus must be prefaced with "**Limits.**") except **maxint**, **maxnat**, **minint** and **minnat**, which are exported unqualified (this means you can call those entry points directly).

Entry Points	DefaultFW	Default fraction width used in printing using the "put" statement.
	DefaultEW	Default exponent width used in printing using the "put" statement.
	minint	The minimum integer in Turing (exported unqualified).
	maxint	The maximum integer in Turing (exported unqualified).
	minnat	The minimum natural number in Turing (exported unqualified).
	maxnat	The maximum natural number in Turing (exported unqualified).

Real numbers are represented in Turing as:
 $f * (\text{radix} ** e) \text{ or } 0$
where for non-zero f:
 $(1 / \text{radix}) \leq \text{abs } (f) \text{ and } \text{abs } (f) < 1.0$
 $\text{minexp} \leq e \text{ and } e \leq \text{maxexp}.$

Radix	The "radix" (usually 2).
--------------	--------------------------

NumDigits	The number of radix digits in f.
MinExp	"minexp" (the smallest exponent allowed).
MaxExp	"maxexp" (the largest exponent allowed).
GetExp	Function that returns the value of "e".
SetExp	Procedure that sets the value of "e".
Rreb	The relative round-off error bound.

ln natural logarithm function

Syntax	ln (<i>r</i> : real) : real
Description	The ln function is used to find the natural logarithm (base e) of a number. For example, ln (1) is 0.
Example	This program prints out the logarithms of 1, 2, 3, ... up to 100. <pre> for <i>i</i> : 1 .. 100 put "Logarithm of ", <i>i</i>, " is ", ln (<i>i</i>) end for </pre>
Details	See also the exp (exponential) function. You cannot take the logarithm of zero or a negative number.
Note	$\log_n(i) = \ln(i) / \ln(n)$
See also	exp (the exponentiation function). See also predefined unit Math .

locate procedure

Syntax	locate (<i>row</i>, <i>column</i> : int)
--------	---

Description	The locate procedure is used to move the cursor so that the next output from put will be at the given row and column. Row 1 is the top of the screen and column 1 is the left side of the screen.
Example	<p>This program outputs stars of random colors to random locations on the screen. The variable <i>colr</i> is purposely spelled differently from the word <i>color</i> to avoid the procedure of that name (used to set the color of output). The row number is purposely chosen so that it is one less than maxrow. This avoids the scrolling of the screen which occurs when a character is placed in the last column of the last row.</p> <pre> setscreen ("screen") var row, column, colr : int loop randint (row, 1, maxrow - 1) randint (column, 1, maxcol) randint (colr, 0, maxcolor) color (colr) locate (row, column) put "*" .. % Use dot-dot to avoid clearing end of line end loop </pre>
Details	<p>The locate procedure is used to locate the next output based on row and column positions. See also the locatexy procedure which is used to locate the output based x and y positions, where x=0, y=0 is the left bottom of the screen.</p> <p>The screen should be in a "screen" or "graphics" mode. See the setscreen procedure for details. If the screen is not in one of these modes, it will automatically be set to "screen" mode.</p>
See also	<p>setscreen and drawdot.</p> <p>See also predefined unit Text.</p>

locatexy graphics procedure

Pixel graphics only

Syntax	locatexy (<i>x</i> , <i>y</i> : int)
Description	The locatexy procedure is used to move the cursor so that the next output from put will be at approximately (<i>x</i> , <i>y</i>). The exact location may be somewhat to the left of <i>x</i> and below <i>y</i> to force alignment to a character boundary.
Example	This program outputs "Hello" starting at approximately (100, 50) on the screen.


```

setscreen ("graphics")
locatexy ( 100, 50 )
    put "Hello"

```

Details The **locatexy** procedure is used to locate the next output based on x and y positions, where the position x=0, y=0 is the left bottom of the screen. See also the **locate** procedure which is used to locate the output-based row and column positions, where row 1 is the top row and column 1 is the left column.

The screen should be in a "graphics" mode. See the **setscreen** procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.

See also **setscreen** and **drawdot**.
See also predefined unit **Text**.

loop statement

Syntax A *loopStatement* is:

```

loop
    statementsAndDeclarations
end loop

```

Description A **loop** statement causes the statements (and declarations) in it to be repeatedly executed. This continues until terminated by one of its enclosed **exit** statements (or by an enclosed **return** or **result** statement).

Example Output on separate lines: Happy, Happy, Happy, etc.

```

loop
    put "Happy"
end loop

```

Example Read words up to the word Stop.

```

var word : string
loop
    get word
    exit when word = "Stop"
end loop

```

Details	<p>A loop statement can contain more than one exit, or none at all (in which case it is an infinite loop). When the exit when is at the beginning of the loop, the loop works like Pascal's do while; when at the end, the loop works like Pascal's repeat until.</p> <p>Just preceding the statements and declarations, you are allowed to write an "invariant clause" of the form:</p> <p style="text-align: center;">invariant <i>trueFalseExpn</i></p> <p>This clause is equivalent to: assert <i>trueFalseExpn</i>.</p>
---------	---

lower bound

Syntax	lower (<i>reference</i> [, <i>dimension</i>]) : int
Description	The lower attribute is used to find the lower bound of an array, string, char (<i>n</i>) or non-opaque subrange type. Since the lower bound is necessarily known at compile time, lower is rarely used.
See also	upper which finds the upper bound.

Math All

Description	<p>This unit contains all the mathematical routines. There are three routines that are part of the language, but are conceptually part of the Math unit.</p> <p>All routines in the Math unit are exported unqualified. (This means you can call the entry points directly.)</p> <p>Descriptions of all the subprograms in the Math module can be found in this chapter.</p>	
Entry Points	abs *	The absolute value function.
	arctan	The arctan function (radians).
	arctand	The arctan function (degrees).

cos	The cosine function (radians).
cosd	The cosine function (degrees).
exp	The exponentiation function.
ln	The natural logarithm function.
max *	The maximum value function.
min *	The minimum value function.
sign	The sign function.
sin	The sine function (radians).
sind	The sine function (degrees).
sqrt	The square root function.
* Part of the language, conceptually part of the Math unit.	

max maximum function

Syntax	max (<i>expn</i> , <i>expn</i>)
Description	The max function is used to find the maximum of two numbers (the two <i>expn</i> 's). For example, max (5, 7) is 7. If both numbers are int , the result is int . If both numbers are nat (natural numbers), the result is nat . But if one or both of the numbers are real , the result is real . See also the min function.
Example	<p>This program outputs 85.72.</p> <pre> var x : real := 74.61 var y : real := 85.72 put max (x, y) % Outputs 85.72 </pre>
Example	<p>This program inputs 10 numbers and outputs their maximum.</p> <pre> var m, t : real get m % Input first number for i : 2 .. 10 % Handle remaining 9 numbers get t m := max (m, t) end for put "The maximum is ", m </pre>

See also See also predefined unit **Math**.

maxcol maximum column function

Syntax	maxcol : int
Description	The maxcol function is used to determine the number of columns on the screen.
Example	This program outputs the maximum column number. <pre>put "Number of columns on the screen is ", maxrow</pre>
Details	For IBM PC compatibles as well as most UNIX dumb terminals, in "text" or "screen" mode, maxcol = 80. For the default IBM PC compatible "graphics" mode (CGA), maxcol = 40.
See also	locate procedure for an example of the use of maxcol .

maxcolor graphics function

Syntax	maxcolor : int
Description	The maxcolor function is used to determine the maximum color number for the current mode of the screen. The alternate spelling is maxcolour .
Example	This program outputs the maximum color number. <pre>setscreen ("graphics") ... put "The maximum color number is ", maxcolor</pre>
Details	The screen should be in a "screen" or "graphics" mode. If it is not, it will automatically be set to "screen" mode. See setscreen for details. For IBM PC compatibles in "screen" mode, maxcolor = 15. For the default IBM PC compatible "graphics" mode (CGA), maxcolor = 3.

See also **drawdot** and **palette** for examples of the use of **maxcolor**. See the **color** procedure which is used for setting the currently active color.

maxint maximum integer function

Syntax	maxint : int
Description	The maxint function is used to determine the largest integer (int) that can be used in a program.
Example	This program outputs the maximum integer. put "The largest integer that can be used is ", maxint
Details	<p>In current Turing and OOT implementations, int values are stored in 4 bytes, i.e., 32 bits. This determines the maximum int value, which is $2^{31} - 1$, equaling 2147483647.</p> <p>There is an anomaly in computer arithmetic in that the absolute value of the largest negative integer is one larger than maxint. Turing reserves this extra value to represent the uninitialized integer. This value can be computed but any attempt to assign it to an int variable is detected as an overflow. You can use this extra value by using the int4 type instead of int, but this type has no initialization checking.</p>
See also	maxnat and minint . See also OOT predefined unit Math .

maxnat maximum natural number function

Syntax	maxnat : nat
Description	The maxnat function is used to determine the largest natural number that can be used in a program.

Example	This program outputs the maximum natural number. put "The largest natural number that can be used is ", maxnat
Details	In current implementations, natural numbers are stored in 4 bytes, i.e., 32 bits. This determines the maximum natural number, which is $2^{32} - 2$, equaling 4294967294. In four bytes it is possible to represent one more value, namely, $2^{32} - 1 = 4294967295$. This extra value is used in Turing to represent the uninitialized natural number. Although it can be computed, any attempt to assign it to a nat variable is detected as an overflow. You can use this extra value by using the nat4 type instead of nat , but this type has no initialization checking.
See also	maxint and minnat . See also predefined unit Limits .

maxrow maximum row function

Syntax	maxrow : int
Description	The maxrow function is used to determine the number of rows on the screen.
Example	This program outputs the maximum row number. put "Number of rows on the screen is ", maxrow
Details	For IBM PC compatibles, maxrow = 25. For many UNIX dumb terminals, maxrow = 24.
See also	locate procedure for an example of the use of maxrow .

maxx graphics function Pixel graphics only

Syntax	maxx : int
--------	-------------------

Description	The maxx function is used to determine the maximum value of x for the current graphics mode.
Example	This program outputs the maximum x value. <pre> setscreen ("graphics") ... put "The maximum x value is ", maxx </pre>
Details	The screen should be in a " <i>graphics</i> " mode. If it is not, it will automatically be set to " <i>graphics</i> " mode. See setscreen for details. For the default IBM PC compatible graphics mode (CGA), maxx = 319.
See also	drawdot for an example of the use of maxx and for a diagram illustrating x and y positions.

maxy graphics function

Pixel graphics only

Syntax	maxy : int
Description	The maxy function is used to determine the maximum value of y for the current graphics mode.
Example	This program outputs the maximum y value. <pre> setscreen ("graphics") ... put "The maximum y value is ", maxy </pre>
Details	The screen should be in a " <i>graphics</i> " mode;. If it is not, it will automatically be set to " <i>graphics</i> " mode. See setscreen for details. For the default IBM PC compatible graphics mode (CGA), maxy = 199.
See also	drawdot for an example of the use of maxy and for a diagram illustrating x and y positions.

min minimum function

Syntax **min** (*expn* , *expn*)

Description The **min** function is used to find the minimum of two numbers (the two *expn*'s). For example, **min** (5, 7) is 5. If both numbers are **int**, the result is **int**. If both numbers are **nat** (natural numbers), the result is **nat**. But if one or both of the numbers are **real**, the result is **real**. See also the **max** function.

Example This program outputs 74.61.

```
var x : real := 74.61
var y : real := 85.72
      put min ( x, y )           % Outputs 74.61
```

Example This program inputs 10 numbers and outputs their minimum.

```
var m, t : real
get m           % Input first number
for i : 2 .. 10 % Handle remaining 9 numbers
  get t
  m := min ( m, t )
end for
      put "The minimum is ", m
```

See also See also predefined unit **Math**.

minint minimum integer function

Syntax **minint** : **int**

Description The **minint** function is used to determine the smallest integer (**int**) that can be used in a program.

Example This program outputs the maximum integer.

```
      put "The smallest integer that can be used is ", minint
```


Details	<p>In current implementations, int values are stored in 4 bytes, i.e., 32 bits. This determines the minimum int value, which is $-2^{31} - 1$, equaling -2147483647.</p> <p>There is an anomaly in computer arithmetic in that the absolute value of the largest negative integer is one larger than maxint. Turing reserves this extra value to represent the uninitialized integer. This value can be computed but any attempt to assign it to an int variable is detected as an overflow. You can use this extra value by using the int4 type instead of int, but this type has no initialization checking.</p>
See also	<p>minnat and maxint.</p> <p>See also predefined unit Limits.</p>

minnat minimum natural number function

Syntax	minnat : nat
Description	The minnat function is used to determine the smallest natural number that can be used in a program.
Example	<p>This program outputs the smallest natural number.</p> <pre>put "The smallest natural number that can be used is ", minnat</pre>
Details	<p>In current Turing and OOT implementations, natural numbers are stored in 4 bytes, i.e., 32 bits. However, the minimum natural number in all implementations is 0. minnat is provided for purposes of symmetry with minint, maxint and maxnat.</p> <p>In four bytes it is possible to represent one more value, namely, $2^{32} - 1 = 4294967295$. This extra value is used in Turing to represent the uninitialized natural number. Although it can be computed, any attempt to assign it to a nat variable is detected as an overflow. You can use this extra value by using the nat4 type instead of nat, but this type has no initialization checking.</p>
See also	<p>minint and maxnat.</p> <p>See also predefined unit Limits.</p>

mod modulo operator

Syntax **mod**

Description The **mod** (*modulo*) operator produces the modulo of one number with another. In other words, the result is always a number between 0 and the second operand. If both operands are positive, the result is identical to the remainder operator. For example, 7 **mod** 2 produces 1 and -12 **mod** 5 produces 3.

Example In this example, *hours* is the current time. It is moved back and forth by a random amount, but the final result must always be between 1 and 12 (the mod operation produces a number between 0 and 11 and then 0 becomes 12).

```
var hours : int := 12
var hoursPassed : int
put "The time is now ", hours, " o'clock"
loop
  randint (hoursPassed, -12, 12)
  exit when hoursPassed = 0
  if hoursPassed < 0 then
    put hoursPassed, " hours before " ..
  else
    put hoursPassed, " hours later " ..
  end if
  put hours, " o'clock" ..
  hours := (hours + hoursPassed) mod 12
  if hours = 0 then
    hours = 12
  end if
  put " it was ", hours, " o'clock"
end loop
```

Details If the second operand is positive, then the result is always non-negative. Likewise, if the second operand is negative, then the result is always non-positive. If both operands are negative, the result is the same as the remainder operator.

See also *infix operators, precedence of operators* and the **rem** and **div** operators.

module declaration

Syntax	<p>A <i>moduleDeclaration</i> is:</p> <pre>module <i>id</i> [implement <i>implementItem</i>] [implement by <i>implementByItem</i>] [import [var] <i>importItem</i> {, [var] <i>importItem</i> }] [export [<i>howExport</i>] <i>id</i> {,[<i>howExport</i>] <i>id</i> }] <i>statementsAndDeclarations</i> end <i>id</i></pre>
Description	<p>A module declaration creates a package of variables, constants, types, subprograms, etc. The name of the module (<i>id</i>) is given in two places, just after module and just after end. Items declared inside the module can be accessed outside of the module only if they are exported. Items from outside the module that are to be used in the module need to be imported (unless they are predefined or pervasive).</p>
Example	<p>This module implements a stack of strings.</p> <pre>module <i>stack</i> % Implements a LIFO list of strings export <i>push, pop</i> var <i>top</i> : int := 0 var <i>contents</i> : array 1 .. 100 of string procedure <i>push</i> (<i>s</i> : string) <i>top</i> := <i>top</i> + 1 <i>contents</i> (<i>top</i>) := <i>s</i> end <i>push</i> procedure <i>pop</i> (var <i>s</i> : string) <i>s</i> := <i>contents</i> (<i>top</i>) <i>top</i> := <i>top</i> - 1 end <i>pop</i> end <i>stack</i> <i>stack</i> . <i>push</i> ("Harvey") var <i>name</i> : string <i>stack</i> . <i>pop</i> (<i>name</i>) % This sets name to Harvey</pre>

Outside of the *stack* module, the procedures *push* and *pop* can be called using the notation *stack.push* and *stack.pop*. This access is allowed because *push* and *pop* are *exported* from the module. Other items declared in the module (*top* and *contents*) cannot be accessed from outside because they are not exported.

Details In some other programming languages, a module is called a *package*, *cluster* or *object*.

A module declaration is executed (it is initialized) by executing its declarations and statements. For example, the *stack* module is initialized by setting the *top* variable to 0. This initialization executes all the statements and declarations in the module that are not contained in procedures or functions. The initialization is completed before any procedure or function of the module can be called from outside the module. An exported subprogram must not be called until initialization of the module is complete.

A call to an exported procedure or function from outside the module executes the body of that procedure or function (the module is *not* initialized with each such call). See also **monitor** and **class** declarations.

The **import** list gives the names of items declared outside the module that can be accessed inside the module. Since *stack* has no **import** list, it is not allowed to access any names declared outside of it. See also **import** lists. Separately-compiled units that are imported are initialized before the importing unit.

The **export** list is used to implement *information hiding*, which isolates implementation details inside the module. The **export** list gives the names of items declared inside the module that can be used outside the module. For example, *push* and *pop* are exported from *stack*. Each such use of an exported item must be preceded by the module name and a dot, for example, *stack.push*. (See **unqualified** for advice on how to avoid using the prefix "*stack.*"). Names that are not exported, such as *top* and *contents*, cannot be accessed outside the module.

Procedures, functions, variables, constants and types can be exported; modules, monitors or classes cannot be exported.

A class is essentially a template for creating individual modules (objects). See **class** for details. A **monitor** is essentially a module in which only one process can be active at a time. See **monitor** and **process** for details.

The **opaque** keyword is used (only) in export lists to precede exported type names that have declarations in the module. Outside of the module, the type will be distinct from all others types. This means, for example, that if the opaque type is a record, its fields cannot be accessed outside of the module. Opaque types are used to guarantee that certain items are inspected and manipulated in only one place, namely, inside the module. These types are sometimes called *abstract data types*. See also **export** lists, which also describes **unqualified** and **pervasive** exports.

Implement and **implement-by** lists are used to separate a module's interface from its body. This allows only a part of a module (its interface) to be visible to its users (its importers), while hiding its implementation. See **implement** and **implement by** lists.

Example Use an **opaque** type to implement complex arithmetic.

```
module complex
  export opaque value, constant, add,
    ... other operations ...

  type value :
    record
      realPt, imagPt : real
    end record

  function constant (realPt, imagPt: real) : value
    var answer : value
    answer . realPt := realPt
    answer . imagPt := imagPt
    result answer
  end constant

  function add (L, R : value) : value
    var answer : value
    answer . realPt := L . realPt + R . realPt
    answer . imagPt := L . imagPt + R . imagPt
    result answer
  end add

  ... other operations for complex arithmetic go here ...
end complex

var c,d : complex .value :=complex.constant ( 1, 5 )
    % c and d become the complex number (1,5)
var e : complex .value := complex.add (c, d )
    % e becomes the complex number (2,10)
```

Details Module declarations can be nested inside other modules but cannot be nested inside procedures or functions. A module must not contain a **bind** as one of its (outermost) declarations. A **return** statement cannot be used as one of the (outermost) statements in a module.

The syntax of a *moduleDeclaration* presented above has been simplified by leaving out **pre**, **invariant** and **post** clauses; the full syntax is:

```
module id
  [ implement implementItem ]
  [ implement by implementByItem ]
  [ import [ var ] importItem {, [ var ] importItem } ]
  [ export [ howExport ] id {, [ howExport ] id } ]
  [ pre trueFalseExpn ]
  statementsAndDeclarations
  [ invariant trueFalseExpn ]
  statementsAndDeclarations
  [ post trueFalseExpn ]
end id
```

The true/false expression in the **pre** and **post** clauses must be true when initialization reaches each of them. After that, these have no effect. The true/false expression in the **invariant** must be true any time the module is exited (when finishing initialization or when returning from an external call to an exported subprogram) or called (via an exported subprogram). These clauses (**pre**, **post** and **invariant**) are not inherited by expansions. For example, if module *B* inherits *A*, the subprograms of *B* are bound by *B*'s clauses and not by *A*'s.

See also **unit**, **monitor** and **class**. See also **export** list, **import** list, **implement** list, **implement by** list, **inherit** list and **deferred** subprogram.

monitor declaration

Syntax *A monitorDeclaration* is:

```
monitor id
    [ implement implementItem ]
    [ implement by implementByItem ]
    [ import [ var ] importItem
      {, [ var ] importItem } ]
    [ export [ howExport ] id {,[howExport ] id } ]
    statementsAndDeclarations
end id
```

Description A monitor is a special purpose module (see **module**) that is used with concurrent processes (see **process**). At most, one concurrent process (see **process**) can be active in a monitor at a time. This means that a process will be blocked if it calls a monitor that is already active. The process will not be allowed to proceed until the monitor is inactive. The monitor provides *mutually exclusive* access to the monitor's internal data.

Example This monitor controls access to the *count* variable so it can be updated by two processes (the *observer* and the *reporter*) without being corrupted by this concurrent access. Generally, it is not safe to have one process update a variable that other processes are simultaneously accessing. The *observer* process repeatedly increments the *counter* when it observes an event. The *reporter* process repeatedly writes out the number of events that have occurred since the last report, resetting the *counter* to zero.

```
monitor controller
export observe, report
```

```

    var counter : int := 0

    procedure observe
        counter := counter + 1
    end observe

    procedure report (var n : int )
        n := counter
        counter := 0
    end report
end controller

process observer
    loop
        ... observe one event ...
        controller . observe
    end loop
end observer

process reporter
    var n : int
    loop
        controller.report ( n )
        ... report n events ...
    end loop
end reporter

fork observer      % Activate the observer
fork reporter      % Activate the reporter

```

Details A **monitor** is essentially a module in which only one process can be active at a time. See **module** declarations for details about initialization. Initialization is the same for modules and monitors.

A monitor can contain **wait** statements (that put processes to sleep) and **signal** statements (that wake them up again). These statements operate on **condition** variables, which are essentially queues of sleeping processes.

A class is essentially a template for creating individual modules (objects). See **class** for details. If the class declaration is preceded by the keyword **monitor**, the created modules are actually monitors. Monitor classes can only inherit (inherit from) other monitor classes.

The body of a monitor has the same form as that of a module, except that modules, monitors and processes cannot be declared inside monitors, and certain statements (**wait** and **signal**) are allowed in monitors.

Details The syntax of a *monitorDeclaration* presented above has been simplified by leaving out **pre**, **invariant** and **post** clauses. See **module** for an explanation of these extra features. There is also an optional *compileTimeIntegerExpression* in the first line, which is explained below. The full syntax is:

```

monitor id [ : compileTimeIntegerExpn ]
    [ implement implementItem ]
    [ implement by implementByItem ]

```

```

[ import [ var ] importItem {, [ var ] importItem } ]
[ export [ howExport ] id {, [ howExport ] id } ]
[ pre trueFalseExpn ]
statementsAndDeclarations
[ invariant trueFalseExpn ]
statementsAndDeclarations
[ post trueFalseExpn ]
end id

```

If the optional *compileTimeIntegerExpression* is present, this is a *device monitor*. Its exclusive access is enforced by an implementation-dependent trick, such as executing it at a hardware priority level given by the expression. A device monitor is restricted from calling monitors (directly or indirectly). This restriction is imposed to eliminate the possibility of blocking a process with a non-zero hardware priority (as this would inadvertently allow multiple entry into a device monitor). It is the programmer's responsibility to meet this restriction; the compiler will not in general enforce the restriction. The current (1999) implementation ignores this *compileTimeIntegerExpression*.

Details An unexported parameterless procedure in a monitor can be specified to be an *interrupt handling procedure* by specifying a device in its header, using the form:

```
procedure id [ : deviceSpecification ]
```

The *deviceSpecification* is a compile time natural number that designates, to the implementation, the class of interrupts that effectively call this procedure. Interrupt handling procedures cannot be called explicitly within the program.

There are two restrictions that the programmer must follow when using interrupt handling procedures; these restrictions will not necessarily be enforced by the software. The first is that an interrupt handling procedure must not execute a **wait**, either directly or indirectly, by calling another procedure. The second is that the interrupt handling procedure must not directly or indirectly cause an exception, unless the exception will be caught by an exception handler that is activated directly or indirectly by the interrupt handling procedure.

Details Declarations of monitors within monitors are disallowed. This would be redundant anyway, as only one process can be inside the outer monitor, so the inner monitor is guaranteed to be successful.

Declarations of classes within monitors are also disallowed.

Any subprogram declared within a subprogram is now allowed to be assigned to a subprogram variable, nor passed as a parametric subprogram.

See also **unit**, **module** and **class**. See also **export** list, **import** list, **implement** list, **implement by** list and **deferred** subprogram.

Mouse



Description	<p>This unit contains the predefined subprograms that deal with using the mouse in a Turing program. The routines allow you to get the current mouse cursor position, check if a button has been pressed and get the information if it has. There are also routines to hide and show the mouse on systems where it makes sense. (On GUI based systems like the Macintosh, the mouse can't be hidden as it may be needed by other applications running at the same time.)</p> <p>All routines in the Mouse module are exported qualified (and thus must be prefaced with "Mouse.").</p>	
Entry Points	Where	Gets the current location of the mouse cursor and status of the mouse buttons.
	Hide	Makes the mouse cursor disappear.
	Show	Displays the mouse cursor.
	ButtonMoved	Checks to see if a mouse button has been pressed.
	ButtonWait	Gets information about a mouse button being pressed such as where it was pressed, which button was pressed, etc.
	ButtonChoose	Selects the mode for the mouse (either single button mode or multi-button mode).

Mouse.ButtonChoose



Syntax	Mouse.ButtonChoose (<i>choice</i> : string)
Description	<p>The Mouse.ButtonChoose procedure is used to change the mode of the mouse. In Turing, the mouse can either be in "<i>single-button mode</i>" or in "<i>multi-button mode</i>". In "<i>single-button mode</i>" the mouse is treated as a one button mouse. A button is considered pressed when any button is pressed and released only when all buttons have been released.</p> <p>In Turing, the mouse starts in "<i>single-button mode</i>".</p>

The parameter *choice* can be one of "singlebutton", "onebutton" (which switch the mouse into "single-button mode") or "multibutton" (which switches the mouse into "multi-button mode").

Example A program that displays the status of the mouse at the top left corner of the screen.

```

Mouse.ButtonChoose ("multibutton")
var x, y, button, left, middle, right : int
Mouse.Where (x, y, button)
left := button mod 10           % left = 0 or 1
middle := (button - left) mod 100 % middle = 0 or 10
right := button - middle - left % right = 0 or 100
if left = 1 then
    put "left button down"
end if
if middle = 10 then
    put "middle button down"
end if
if right = 100 then
    put "right button down"
end if

```

Details Under DOS, the mouse does not start initialized. When the first mouse command is received (**Mouse.Hide**, **Mouse.Show**, **Mouse.Where**, **Mouse.ButtonMoved**, **Mouse.ButtonWait**, **Mouse.ButtonChoose**) Turing attempts to initialize the mouse. If successful, the mouse cursor will appear at that time.

Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the **autoexec.bat** or **config.sys** files).

Status Exported qualified.

This means that you can only call the function by calling **Mouse.ButtonChoose**, not by calling **ButtonChoose**.

See also **Mouse.ButtonMoved** and **Mouse.ButtonWait** to get mouse events saved in a queue. See also **Mouse.Where** to get the current status of mouse button(s).

Mouse.ButtonMoved



Syntax **Mouse.ButtonMoved** (*motion* : **string**) : **boolean**

- Description The **Mouse.ButtonMoved** function indicates whether there is a mouse event of the appropriate type on the mouse queue. Events are either "up", "down", "updown" or "downup" events (although the "downup" and "updown" are the same event).
- The parameter *motion* must be one of "up", "down", "updown" or "downup". If an event of the type requested is in the queue, **Mouse.ButtonMoved** returns **true**. If the event is not in the queue, then **Mouse.ButtonMoved** returns **false**.
- In "single-button mode" (where the mouse is treated like a one-button mouse), a "down" event occurs whenever all the buttons are up and a button is pressed. An "up" event takes place when the last button is released so that no buttons remain pressed.
- In "multi-button mode", a "down" event occurs whenever any button is pressed, and an "up" event occurs whenever any button is released.
- Example This program draws random circles on the screen until the user clicks the mouse button, whereupon it starts drawing random boxes. Clicking the mouse button switches between the two.

```

var circles: boolean := true
loop
  var x, y, radius, clr: int
  if Mouse.ButtonMoved ("down") then
    var buttonnumber, buttonupdown: int
    Mouse.ButtonWait ("down", x, y, buttonnumber,
                      buttonupdown)
    circles := not circles
  end if
  x := Rand.Int (0, maxx)
  y := Rand.Int (0, maxy)
  radius := Rand.Int (0, 100)
  clr := Rand.Int (0, maxcolor)
  if circles then
    Draw.FillOval (x, y, radius, radius, clr)
  else
    Draw.FillBox (x, y, x + radius, y + radius, clr)
  end if
end loop

```

- Example This is an example demonstrating how to check for both character and mouse input at the same time.

```

var ch: string (1)
var x, y, btnnum, btnupdown: int
loop
  if hasch then
    getch (ch)
    Text.Locate (1, 1)
    put "The character entered is a: ", ch
  end if
  if Mouse.ButtonMoved ("down") then
    Mouse.ButtonWait ("down", x, y, btnnum, btnupdown)
    Text.Locate (1, 1)
    put "The button was clicked at position: ", x, ", ", y
  end if
end loop

```

end if

end loop

Details	<p>Mouse.ButtonMoved can be thought of as the mouse equivalent of hasch in that they both check for something in a queue and both return immediately.</p> <p>Under DOS, the mouse does not start initialized. When the first mouse command is received (Mouse.Hide, Mouse.Show, Mouse.Where, Mouse.ButtonMoved, Mouse.ButtonWait, Mouse.ButtonChoose), Turing attempts to initialize the mouse. If the initialization is successful, the mouse cursor will appear.</p> <p>Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the autoexec.bat or config.sys files).</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Mouse.ButtonMoved, not by calling ButtonMoved.</p>
See also	<p>Mouse.ButtonMoved to get mouse events saved in the queue. See also Mouse.ButtonChoose to switch between "single-button mode" and "multi-button mode".</p>

Mouse.ButtonWait



Syntax	Mouse.ButtonWait (<i>motion</i> : string , var <i>x</i> , <i>y</i> , <i>buttonNumber</i> , <i>buttonUpDown</i> : int)
Description	<p>The Mouse.ButtonWait procedure gets information about a mouse event and removes it from the queue.</p> <p>The parameter <i>motion</i> must be one of "up", "down", "updown" or "downup". If an event of the type requested is in the queue, Mouse.ButtonWait returns instantly. If there isn't such an event, Mouse.ButtonWait waits until there is one and then returns (much like getch handles keystrokes).</p> <p>In "single-button mode" (where the mouse is treated like a one-button mouse), a "down" event occurs whenever all the buttons are up and a button is pressed. An "up" event takes place when the last button is released so that no buttons remain pressed.</p> <p>In "multi-button mode", a "down" event occurs whenever any button is pressed, and an "up" event occurs whenever any button is released.</p>

The parameters *x* and *y* are set to the position of the mouse cursor when the button was pressed. The parameter *buttonnumber* is set to 1 when in "single-button mode". In "multi-button mode", it is set to 1 if the left button was pressed, 2 if the middle button was pressed, and 3 if the right button was pressed. The parameter *buttonupdown* is set to 1, if a button was pressed and 0 if a button was released.

Example This program draws lines. It starts a line where the user presses down and continues to update the line while the mouse button is held down. When the button is released, the line is permanently drawn and the user can draw another line.

```
var x, y, btnNumber, btnUpDown, buttons : int
var nx, ny : int
loop
  Mouse.ButtonWait ("down", x, y, btnNumber, btnUpDown)
  nx := x
  ny := y
  loop
    Draw.Line (x, y, nx, ny, 0) % Erase previous line
    exit when Mouse.ButtonMoved ("up")
    Mouse.Where (nx, ny, buttons)
    Draw.Line (x, y, nx, ny, 1) % Draw line to position
  end loop
  Mouse.ButtonWait ("up", nx, ny, btnNumber, btnUpDown)
  Draw.Line (x, y, nx, ny, 2) % Draw line to final position
end loop
```

Example This is an example demonstrating how to check for both character and mouse input at the same time.

```
var ch : string (1)
var x, y, btnNum, btnUpDown : int
loop
  if hasch then
    getch (ch)
    Text.Locate (1, 1)
    put "The character entered is a: ", ch
  end if
  if Mouse.ButtonMoved ("down") then
    Mouse.ButtonWait ("down", x, y, btnNum, btnUpDown)
    Text.Locate (1, 1)
    put "The button was clicked at position: ", x, ", ", y
  end if
end loop
```

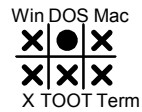
Details **Mouse.ButtonWait** can be thought of as the mouse equivalent of **getch** in that they both read something in a queue and both wait until they get the thing they're looking for.

Under DOS, the mouse does not start initialized. When the first mouse command is received (**Mouse.Hide**, **Mouse.Show**, **Mouse.Where**, **Mouse.ButtonMoved**, **MouseButtonWait**, **Mouse.ButtonChoose**), Turing attempts to initialize the mouse. If the initialization is successful, the mouse cursor will appear.

Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the **autoexec.bat** or **config.sys** files).

Status	Exported qualified. This means that you can only call the function by calling Mouse.ButtonWait , not by calling ButtonWait .
See also	Mouse.ButtonWait to see if an appropriate event is in the queue. See also Mouse.ButtonChoose to switch between " <i>single-button mode</i> " and " <i>multi-button mode</i> ".

Mouse.Hide



Syntax	Mouse.Hide
Description	This function causes the mouse cursor to disappear. Mouse.Show makes the mouse cursor re-appear. While the cursor is hidden, the program still gets button events and Mouse.Where still functions normally. Mouse.Hide has no effect on systems with Windows (Macintosh, X Windows, MS Windows).
Example	<p>A program that hides the mouse cursor whenever it is outside the box (100, 100) - (200, 200).</p> <pre>var x, y, button : int loop Mouse.Where (x, y, button) if x < 100 or x > 200 or y < 100 or y > 200 then Mouse.Hide else Mouse.Show end if end loop</pre>
Details	<p>Under DOS, the mouse does not start initialized. When the first mouse command is received (Mouse.Hide, Mouse.Show, Mouse.Where, Mouse.ButtonMoved, Mouse.ButtonWait, Mouse.ButtonChoose), Turing attempts to initialize the mouse. If the initialization is successful, the mouse cursor will appear.</p> <p>Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the autoexec.bat or config.sys files).</p>

Status	Exported qualified. This means that you can only call the function by calling Mouse.Hide , not by calling Hide .
See also	Mouse.Show to make the mouse cursor re-appear.

Mouse.Show



Syntax	Mouse.Show
Description	This function causes the mouse cursor to appear. Mouse.Hide makes the mouse cursor disappear. Mouse.Show has no effect on systems with Windows (Macintosh, X Windows, MS Windows).
Example	<p>A program that hides the mouse cursor whenever it is outside the box (100, 100) - (200, 200).</p> <pre> var x, y, button : int loop Mouse.Where (x, y, button) if x < 100 or x > 200 or y < 100 or y > 200 then Mouse.Hide else Mouse.Show end if end loop </pre>
Details	<p>Under DOS, the mouse does not start initialized. When the first mouse command is received (Mouse.Hide, Mouse.Show, Mouse.Where, Mouse.ButtonMoved, Mouse.ButtonWait, Mouse.ButtonChoose), Turing attempts to initialize the mouse. If the initialization is successful, the mouse cursor will appear.</p> <p>Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the autoexec.bat or config.sys files).</p>
Status	Exported qualified. This means that you can only call the function by calling Mouse.Show , not by calling Show .
See also	Mouse.Hide to make the mouse cursor disappear.

Mouse.Where



Syntax	Mouse.Where (var <i>x</i> , <i>y</i> , <i>button</i> : int)
Description	<p>This Mouse.Where procedure is used to get current information about the status of the mouse. The parameters <i>x</i> and <i>y</i> are set to the current location of the mouse cursor. If the program is running on a system using windows, the cursor may be outside the window. This means that <i>x</i> and <i>y</i> may be set to values outside of the bounds of 0 to maxx and 0 to maxy. When running under DOS, <i>x</i>, <i>y</i> and <i>button</i> are set to -1 if there is no mouse available.</p> <p>The parameter <i>button</i> is set depending on the current mode. In "<i>single-button mode</i>" (where the mouse is treated like a one-button mouse), <i>button</i> is set to 0 if all the mouse buttons are up, and 1 if any of the mouse buttons are down. In "<i>multi-button mode</i>", <i>button</i> is assigned the sum of 1 if the left button is down, 10 if the middle button is down, and 100 if the right button is down. Thus if <i>button</i> has the value of 101, then it means that the left and right mouse buttons were depressed.</p>
Example	<p>A program that displays the status of the mouse at the top left corner of the screen.</p> <pre>var <i>x</i>, <i>y</i>, <i>button</i> : int loop Mouse.Where (<i>x</i>, <i>y</i>, <i>button</i>) Text.Locate (1, 1) if <i>button</i> = 0 then put <i>x</i> : 4, " ", <i>y</i> : 4, " button up" else put <i>x</i> : 4, " ", <i>y</i> : 4, " button down" end if end loop</pre>
Details	<p>Under DOS, the mouse does not start initialized. When the first mouse command is received (Mouse.Hide, Mouse.Show, Mouse.Where, Mouse.ButtonMoved, Mouse.ButtonWait, Mouse.ButtonChoose) Turing attempts to initialize the mouse. If the initialization is successful, the mouse cursor will appear.</p> <p>Note that under DOS, there must be both a mouse attached and a mouse driver (usually found in the autoexec.bat or config.sys files).</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Mouse.Where, not by calling Where.</p>

See also **Mouse.ButtonMoved** and **Mouse.ButtonWait** to get mouse events saved in a queue. See also **Mouse.ButtonChoose** to switch between "*single-button mode*" and "*multi-button mode*".

Music

Description This unit contains the predefined subprograms that deal with sound and music. Some of these routines have not been implemented at the time of the writing of this manual and will be implemented in future releases. All routines in the **Music** module are exported qualified (and thus must be prefaced with "**Music.**").

Entry Points

Play	Plays a series of notes.
PlayFile	Plays music from a file. File must be in an allowable format.
Sound	Plays a specified frequency for a specified duration.
SoundOff	Immediately terminates any sound playing.

Music.Play



Syntax **Music.Play** (*music* : **string**)

Description The **Music.Play** procedure is used to sound musical notes on the computer. Sounds are produced synchronously on a per process basis. This means that when a process executes a **Music.Sound** or **Music.Play** command, it stops until the command is finished. However, other processes will continue to executing.

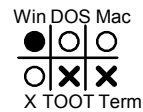
Example This program sounds the first three notes of the C scale.
Music.Play ("cde")

Example This program plays from middle C to one octave above middle C and down again in 8th notes.

```
Music.Play ( "8cdefgab>c" )
      Music.Play ( "<bagfedc" )
```

Details	<p>The syntax of the play string may be enhanced in the future.</p> <p>The Music.Play procedure takes strings containing characters that specify notes, rests, sharps, flats and duration. The notes are the letters a to g (or A to G). A rest is p (for pause). A sharp is + and a flat is -. The durations are 1 (whole note), 2 (half note), 4 (quarter note), 8 (eight note) and 6 (sixteenth note). The character > raises to the next octave and < lowers. For example, this is the way to play C and then C sharp one octave above middle C with a rest between them, all in sixteenth notes: Music.Play(">6cpc+"). Blanks can be used for readability and are ignored by Music.Play.</p> <p>Under some systems such as UNIX, the Music.Play procedure may have no effect.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Music.Play, not by calling Play.</p>
See also	<p>the Music.Sound procedure, which makes a sound of a given frequency (Hertz) and duration (milliseconds).</p>

Music.PlayFile



Syntax	Music.PlayFile (<i>fileName</i> : string)						
Description	<p>The Music.PlayFile procedure is used to play a file of music. The file must be in one of the acceptable formats and the machine, must have the appropriate hardware.</p> <p>The <i>fileName</i> parameter must give the format of the file:</p> <table> <tr> <td>WAV files</td><td>"WAV:filename" or "filename.WAV"</td></tr> <tr> <td>SND files</td><td>"SND:filename" or "filename.SND"</td></tr> <tr> <td>MOD files</td><td>"MOD:filename" or "filename.MOD"</td></tr> </table> <p>Sounds are produced synchronously on a per process basis. This means that when a process executes a Music.Sound, Music.Play or Music.PlayFile command, it stops until the command is finished. However, other processes will continue executing.</p>	WAV files	"WAV:filename" or "filename.WAV"	SND files	"SND:filename" or "filename.SND"	MOD files	"MOD:filename" or "filename.MOD"
WAV files	"WAV:filename" or "filename.WAV"						
SND files	"SND:filename" or "filename.SND"						
MOD files	"MOD:filename" or "filename.MOD"						
Example	<p>This program plays the music in the file "branden3.wav" while drawing ovals on the screen.</p>						

```

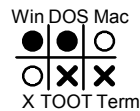
process DoMusic
  loop
    Music.PlayFile ( "branden3.wav" )
  end loop
end DoMusic

fork DoMusic
var x, y, clr : int
loop
  x := Rand.Int (0, maxx)
  y := Rand.Int (0, maxy)
  clr := Rand.Int (0, maxcolor)
  Draw.FillOval (x, y, 30, 30, clr)
end loop

```

Details	<p>The file formats that each system can play will vary. As well, the correct hardware must be available. The release notes document which format each system can handle.</p> <p>At the time of writing, this subprogram was not supported on any platform. Check the release notes to see which file types are supported in the current version.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Music.PlayFile, not by calling PlayFile.</p>

Music.Sound



Syntax	Music.Sound (<i>frequency</i> , <i>duration</i> : int)
Description	<p>The Music.Sound statement is used to cause the computer to sound a note of a given frequency for a given time. The frequency is in cycles per second (Hertz). The time duration is in milliseconds. For example, middle A on a piano is 440 Hertz, so Music.Sound(440, 1000) plays middle A for one second.</p> <p>Sounds are produced synchronously on a per process basis. This means that when a process executes a Music.Sound or Music.Play command, it stops until the command is finished. However, other processes will continue executing.</p>
Example	<p>This program plays a siren sound in the background.</p> <pre> process siren loop </pre>

```

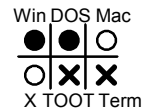
    for i : 100 .. 3000 by 100
        Music.Sound ( i, 50 ) % Sound note
    end for
    for decreasing i : 2900 .. 200 by 100
        Music.Sound ( i, 50 ) % Sound note
    end for
end loop
end siren

fork siren
    ... the rest of the program goes here while the siren continues ...

```

Details	On IBM PC compatibles, the hardware resolution of duration is in units of 55 milliseconds. For example, Music.Sound (440, 500) will delay the program by about half a second, but may be off by as much as 55 milliseconds.
Status	Exported qualified. This means that you can only call the function by calling Music.Sound , not by calling Sound .
See also	Music.Play statement, which plays notes based on musical notation. For example, Music.Play ("8C") plays an eighth note of middle C.

Music.SoundOff



Syntax	Music.SoundOff
Description	The Music.SoundOff procedure stops any sound or music that is currently playing or is waiting to play.
Status	Exported qualified. This means that you can only call the function by calling Music.SoundOff , not by calling SoundOff .
See also	Music.Play , Music.PlayFile , and Music.Sound procedures, which make sounds that can be turned off with Music.SoundOff .

named type

Syntax	<p>A <i>namedType</i> is one of:</p> <ul style="list-style-type: none">(a) <i>typeId</i>(b) <i>moduleId . typeId</i>
Description	A type can be given a name (<i>typeId</i>) and later this name can be used instead of writing out the type.
Example	<p>In this example, <i>phoneRecord</i> is a named type.</p> <pre>type phoneRecord : record name : string (20) phoneNumber : int address : string (50) end record ... var oneEntry : phoneRecord var phoneBook : array 1 .. 100 of phoneRecord</pre>
Details	<p>Form (a) is the most common kind of named type. Form (b) is used when the type name has been exported from a module.</p> <p>Arrays whose bounds are not known at compile time cannot be named.</p>

nargs number of arguments

Syntax	nargs : int
Description	<p>The nargs function is used to determine the number of arguments that have been passed to a program from the command line. For example, if the program is run from the Turing environment using</p> <pre>:r file1 file2</pre> <p>then nargs will return 2. If a program called <i>prog.x</i> is run under UNIX using this command:</p> <pre>prog.x file1 file2</pre>

the value of **nargs** will similarly be 2.

The **nargs** function is usually used together with the **fetcharg** function to access the arguments that have been passed to the program.

See also **fetcharg** for an example of the use of **nargs**.

nat natural number type

Syntax **nat**

Description The **nat** (natural number) type has the values 0, 1, 2, 3 ... Natural numbers can be combined by various operators, such as addition (+) and multiplication (*). Natural numbers can be combined with integers (type **int**), in which case the result is an integer. Natural numbers can also be combined with **real** numbers, in which case the result is generally a real number. Natural numbers can always be assigned to real variables, with implicit conversion to **real**.

Example

```
var counter : nat  
             var j : nat := 9
```

Details See also *explicitIntegerConstant*. The **nat** type is used instead of **int** when the values are known to be non-negative.

The Turing operators on natural numbers are the same as those for integers: +, -, * (multiply), **div** (truncating integer division), **mod** (integer remainder), ** (exponentiation), as well as comparisons (+, **not=**, >, >=, <, <=). The operators **and**, **or** and **xor** to be applied to natural number values. The bit-wise boolean result is produced as a natural number. The **shr** (shift right) and **shl** (shift left) operators are also introduced.

In the current implementation, the range of natural numbers is from 0 to 4294967294. In other words, the maximum value of a natural number is 2**32 - 2. This range exists because natural numbers are stored in 4 bytes. The types **nat1**, **nat2** and **nat4** specify natural numbers that fit into 1, 2 or 4 bytes.

Explicit constants such as 213 and 0 are considered to be integers. As a result the type of *tax* in this declaration is **int**:

```
var tax := 0    % The type is int
```

Natural number values can be used whenever integer values are expected and vice versa, given that the value does not exceed the range of the expected type.

When integer and natural numbers are combined using a binary operator such as `+`, the result is an integer. This means, for example, that if *counter* is a natural number, *counter* + 1 is considered to be an integer. As long as the result fits into the range that is the intersection of the ranges of **int** and **nat**, the result will be as expected. Anomalies occur when the result is (or would be) greater than the largest integer (**maxint**=2147483647). For example, if natural number *n* is greater than **maxint**, the expression *n* + 1 will overflow, because its result is an **int** (because 1 is an **int**). To avoid this problem, you must be careful that both operands are natural numbers. Suppose we have this declaration:

```
const natOne : nat := 1
```

We can safely compute *n* + *natOne* because both operands have type **nat**.

Natural numbers can be converted to real numbers using **natreal**, but in practice this is rarely used, because a natural value used in place of a real value will be automatically converted to real.

Natural numbers can be converted to strings and back using **natstr** and **strnat**.

In the C language, a natural number is said to be "unsigned".

See also **maxnat**, **int**, **natn**, **intn**, **natstr**, **strnat** and **natreal**.

natn n-byte natural number type

Dirty

Syntax

- (a) **nat1** % 1-byte natural number
- (b) **nat2** % 2-byte natural number
- (c) **nat4** % 4-byte natural number

Description The **natn** (*n*-byte natural number) types are machine-dependent types that occupy a specified number of bytes. By contrast, the **nat** type is in principle a machine-independent and mathematical type (it overflows, however, when the value is too large or small, that is, when the value does not fit into 4 bytes).

Example

```
var counter1 : nat1            % Range is 0 .. 255
var counter2 : nat2            % Range is 0 .. 65536
var counter4 : nat4            % Range is 0 .. 4294967295
```

Details	<p>In Turing, the range of the nat is 0 to 4294967294, which means that the nat4 type allows one more value, 4294967295. This extra value is used in nat to represent the state of being uninitialized. The natn types allow use of all possible values that fit into n bytes and thereby eliminates checking for initialization.</p> <p>The natn types are like the C language types <i>short unsigned</i>, <i>unsigned</i>, and <i>long unsigned</i>, except that the number of bytes occupied by the C types depends on the particular C compiler.</p>
See also	the intn types which are n byte integer values. See also nat and int .

natreal natural number to real function

Syntax	natreal ($n : \mathbf{nat}$) : real
Description	<p>The natreal function is used to convert a natural number to a real number. This function is rarely used, because in Turing, a natural number can be used anyplace a real value is required. When this is done, the natreal function is implicitly called to do the conversion from nat to real. The natreal function is similar to intreal, except that natreal handles values that are larger than int values and does not handle negative values.</p>
See also	nat . See also the intreal , floor , ceil and round functions.

natstr natural-number-to-string function

Syntax	natstr ($n : \mathbf{nat}$ [, $width : \mathbf{int}$ [, $base : \mathbf{int}$]]) : string
Description	<p>The natstr function is used to convert a natural number to a string. The string is equivalent to n, padded on the left with blanks as necessary to a length of $width$, written in the given number $base$. For example, natstr (14, 4, 10) = "bb14" where b represents a blank. The $width$ and $base$ parameters are both optional. If they are omitted, the string is made just long enough to hold the value and the number base is 10. For example, natstr (23) = "23".</p>

The *width* parameter must be non-negative. If *width* is not large enough to represent the value of *i*, the length is automatically increased as needed.

The string returned by **natstr** is of the form:

{blank}digit{digits}

where {blank} means zero or more blanks and digit{digit} means one or more digits. The leftmost digit is either non-zero, or a single zero digit; in other words, leading zeros are suppressed.

The letters A, B, C ... are used to represent the digit values 10, 11, 12, ... The *base* must be in the range 2 to 36 (36 because there are ten digits and 26 letters). For example, **natstr** (255, 0, 16) = "FF".

The **natstr** function is the inverse of **strnat**, so for any natural number *n*, **strnat** (**natstr**(*n*)) = *n*.

See also **chr**, **ord** and **strnat** functions. See also the **intstr** and **strint** functions. See also *explicitIntegerConstant* for the way to write values in base 2 and base 16 in a program.



Description The Net module allows TCP/IP equipped machines to communicate. In the current implementation (WinOOT 3.0), this is available only under Win32 (Windows 95, 98, NT and later).

To allow two machines to communicate, there must be a server (which calls **Net.WaitForConnection**) and a client (which calls **Net.OpenConnection**). The server waits until a client connects and then starts communication between the two. When a connection is established, a net stream is returned that can be used in the same fashion as a file stream (i.e. using **puts** and **gets**). Once the connection is finished, the programs call **Net.CloseConnection**.

For ease of reading web pages, the **Net.OpenURLConnection** opens up a URL for reading with the **get** statement. It is up to the user program to interpret the HTML or file located at the URL.

All subprograms in the **Net** unit are exported qualified (and thus must be prefaced with "**Net.**").

Entry Points **WaitForConnection** Waits until a client connects to a specified port.

OpenConnection Opens a connection to a specified machine.

OpenURLConnection Opens a connection to a file specified by a URL.

CloseConnection Closes a specified connection.

BytesAvailable Returns the number of bytes available to be read from a net stream.

CharAvailable Returns true if there is a character available to be read from a net stream.

LineAvailable Returns true if there is a line of text available to be read from a net stream.

TokenAvailable Returns true if there is a token available to be read from a net stream.

HostAddressFromName Returns a host's address given its host name.

HostNameFromAddress Returns a host's name given its address.

LocalAddress Returns the host name of the local machine.

LocalName Returns the TCP/IP address of the local machine.

Net.BytesAvailable



Syntax	Net.BytesAvailable (<i>netStream</i> : int) : int
Description	Returns the number of bytes available for reading from the net stream specified by the <i>netStream</i> parameter.
Details	<p>The Net module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols</p> <p>It is possible for Firewalls to interfere with the actions of the Net module, preventing connections from taking place.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Net.BytesAvailable, not by calling BytesAvailable.</p>
See also	Net.CharAvailable , Net.LineAvailable , and Net.TokenAvailable .

Net.CharAvailable



Syntax	Net.CharAvailable (<i>netStream</i> : int) : boolean
Description	Returns true if a character is waiting to be read from the net stream specified by the <i>netStream</i> parameter. If Net.CharAvailable returns true , then a single character can be read from the stream without blocking.
Details	<p>The Net module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols</p> <p>It is possible for Firewalls to interfere with the actions of the Net module, preventing connections from taking place.</p>
Example	<p>The following program fragment reads a character from <i>netStream</i> only if there is one waiting to be read.</p> <pre>if Net.CharAvailable (netStream) then var ch : char get : netStream, ch put ch .. end if</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Net.CharAvailable, not by calling CharAvailable.</p>
See also	Net.BytesAvailable , Net.LineAvailable , and Net.TokenAvailable .

Net.CloseConnection



Syntax	Net.CloseConnection (<i>netStream</i> : int)
Description	Closes a network connection made with Net.OpenConnection or Net.WaitForConnection . After the connection is closed, the net stream cannot be used for any purpose on either side of the connection.
Details	The Net module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols

It is possible for Firewalls to interfere with the actions of the **Net** module, preventing connections from taking place.

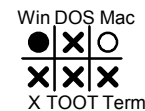
Example The following program fragment connects to port 5300 on the machine specified by *netAddress*, sends the work OK to it and closes the connection.

```
netStream := Net.OpenConnection (netAddress, chatPort)
if netStream <= 0 then
    put "Unable to connect to ", netAddress
    return
end if
put : netStream, "OK"
    Net.CloseConnection (netStream)
```

Status Exported qualified.
This means that you can only call the function by calling **Net.CloseConnection**, not by calling **CloseConnection**.

See also **Net.OpenConnection** and **Net.WaitForConnection**.

Net.HostAddressFromName



Syntax **Net.HostAddressFromName** (
 hostName : **string**) : **string**

Description Returns the numeric TCP/IP address of the machine whose hostname is specified by the *hostName* parameter.

Details The **Net** module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols
It is possible for Firewalls to interfere with the actions of the **Net** module, preventing connections from taking place.

Example The following program prints out the hostname of the current machine.

```
var hostName : string := "www.holtsoft.com"
put "The machine address of ", hostName, " is ",
    Net.HostAddressFromName (hostName)
```

Status Exported qualified.
This means that you can only call the function by calling **Net.HostAddressFromName**, not by calling **HostAddressFromName**.

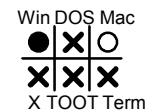
See also **Net.HostNameFromAddress**.

Net.HostNameFromAddress



Syntax	Net.HostNameFromAddress (<i>hostAddr</i> : string) : string
Description	Returns the TCP/IP hostname of the machine whose numeric address is specified by the <i>hostAddr</i> parameter.
Details	The Net module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols It is possible for Firewalls to interfere with the actions of the Net module, preventing connections from taking place.
Example	The following program prints out the hostname of the machine whose TCP/IP numeric address is "128.100.5.1". <pre>var hostAddr : string := "128.100.5.1" put "The machine name of ", hostAddr, " is ", Net.HostNameFromAddress (hostAddr)</pre>
Status	Exported qualified. This means that you can only call the function by calling Net.HostNameFromAddress , not by calling LocalName .
See also	Net.HostAddressFromName .

Net.LineAvailable



Syntax	Net.LineAvailable (<i>netStream</i> : int) : boolean
Description	Returns true if a line of input is waiting to be read from the net stream specified by the <i>netStream</i> parameter. If Net.LineAvailable returns true , then a line of input can be read from the stream without blocking.
Details	The Net module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols

It is possible for Firewalls to interfere with the actions of the **Net** module, preventing connections from taking place.

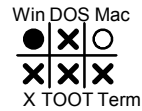
Example The following program fragment reads a character from *netStream* only if there is one waiting to be read.

```
if Net.LineAvailable (netStream) then
  var line : string
  get : netStream, line : *
  put line
end if
```

Status Exported qualified.
This means that you can only call the function by calling **Net.LineAvailable**, not by calling **LineAvailable**.

See also **Net.BytesAvailable**, **Net.CharAvailable**, and **Net.TokenAvailable**.

Net.LocalAddress



Syntax **Net.LocalAddress : string**

Description Returns the TCP/IP numeric address of the machine the program is running on. The numeric address is of the form *xxx.yyy.zzz.www* where each segment is a number from 0 to 255.

Details The **Net** module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols
It is possible for Firewalls to interfere with the actions of the **Net** module, preventing connections from taking place.

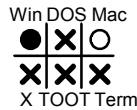
Example The following program prints out the TCP/IP numeric address of the current machine.

```
put "Your machine address is ", Net.LocalAddress
```

Status Exported qualified.
This means that you can only call the function by calling **Net.LocalAddress**, not by calling **LocalAddress**.

See also **Net.LocalName**.

Net.LocalName



Syntax	Net.LocalName : string
Description	Returns the TCP/IP hostname of the machine the program is running on.
Details	<p>The Net module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols</p> <p>It is possible for Firewalls to interfere with the actions of the Net module, preventing connections from taking place.</p>
Example	<p>The following program prints out the hostname of the current machine.</p> <pre>put "Your machine name is ", Net.LocalName</pre>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Net.LocalName, not by calling LocalName.</p>
See also	Net.LocalAddress .

Net.OpenConnection



Syntax	Net.OpenConnection (<i>netAddr</i> : string, port : int) : int
Description	<p>Attempts to open a connection to port specified by the <i>port</i> parameter on the machine specified by <i>netAddr</i> parameter. There must be a program listening to that port for the connection to be made. In OOT, this is done using the Net.WaitForConnection function.</p> <p>If successful, Net.OpenConnection returns a network stream descriptor which can be used with the put, get, read, and write statements and eof function to send and receive data to the listening program. It is also the parameter used for the Net.CloseConnection, Net.BytesAvailable, Net.CharAvailable, Net.LineAvailable, and Net.TokenAvailable functions.</p>

The *netAddr* parameter is a string specifying the net address of the machine to be connected to. This can either be the full hostname or the numerical address.

In general, system program listen in on ports with numbers below 1024. Port numbers above 1024 are generally available for use by user created programs.

The program will wait for an indeterminate amount of time to make the connection. If it fails, it will return a non-positive value.

- Details The **Net** module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols
- It is possible for Firewalls to interfere with the actions of the **Net** module, preventing connections from taking place.
- Example The following program implements a "Chat" program. One user runs the program on their machine as a server, which waits for another machine to connect to it. The second user specifies the machine to connect to and then connects. The two can then type at each other.

```
% The "Chat" program
const chatPort : int := 5055
var choice : int
loop
  put "Enter 1 to run chat server"
  put "Enter 2 to run chat session"
  put "Choice: " ..
  get choice
  exit when choice = 1 or choice = 2
end loop

var netStream : int
var netAddress : string

if choice = 1 then
  netStream := Net.WaitForConnection (chatPort, netAddress)
else
  put "Enter the address to connect to: " ..
  get netAddress
  netStream := Net.OpenConnection (netAddress, chatPort)
  if netStream <= 0 then
    put "Unable to connect to ", netAddress
    return
  end if
end if
Draw.Cls
put "Connected to ", netAddress

var localRow : int := 2
var localCol : int := 1
var remoteRow := maxrow div 2
var remoteCol : int := 1
var ch : char

View.Set ("noecho")
```



```

loop
  if hasch then
    ch := getchar
    put : netStream, ch
    if ch = '\n' then
      localRow := localRow mod (maxrow div 2) + 1
      localCol := 1
      Text.Locate (localRow, localCol)
      put "" % Clear to end of line
      Text.Locate (localRow, localCol)
    else
      Text.Locate (localRow, localCol)
      put ch ..
      localCol += 1
    end if
  end if
end if

if Net.CharAvailable (netStream) then
  get : netStream, ch
  if ch = '\n' then
    remoteRow := remoteRow mod (maxrow div 2) +
      1 + (maxrow div 2)
    remoteCol := 1
    Text.Locate (remoteRow, remoteCol)
    put "" % Clear to end of line
    Text.Locate (remoteRow, remoteCol)
  else
    Text.Locate (remoteRow, remoteCol)
    put ch ..
    remoteCol += 1
  end if
end if
end loop

```

Status Exported qualified.
 This means that you can only call the function by calling
 Net.OpenURLConnection, not by calling **OpenURLConnection**.

See also **Net.WaitForConnection** and **Net.CloseConnection**.

Net.OpenURLConnection

Syntax **Net.OpenURLConnection (urlAddr : string) : int**

Description Attempts to open a http connection to pthe URL (Universal Resource
 Locator) specified by the urlAddr.

If successful, **Net.OpenURLConnection** returns a network stream descriptor which can be used with the **get** statement and **eof** function to read the web page located at the URL.

The program will wait for an indeterminate amount of time to make the connection. If it fails, it will return a non-positive value.

Details The **Net** module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols

It is possible for Firewalls to interfere with the actions of the **Net** module, preventing connections from taking place.

Example The following program prints out the contents of the file specified by the user.

```
var url : string
put "Enter the URL to load: " ..
get url

var netStream : int
var line : string

netStream := Net.OpenURLConnection (url)
if netStream <= 0 then
    put "Unable to connect to ", url
    return
end if
loop
    exit when eof (netStream)
    get : netStream, line
    put line
end loop

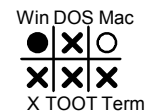
Net.CloseConnection (netStream)
```

Status Exported qualified.

This means that you can only call the function by calling **Net.OpenURLConnection**, not by calling **OpenURLConnection**.

See also **Net.CloseConnection**.

Net.TokenAvailable



Syntax **Net.TokenAvailable (netStream : int) : boolean**

Description	Returns true if a line of input is waiting to be read from the net stream specified by the <i>netStream</i> parameter. If Net.TokenAvailable returns true , then a single token (character surrounded by whitespace) can be read from the stream without blocking.
Details	The Net module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols It is possible for Firewalls to interfere with the actions of the Net module, preventing connections from taking place.
Example	The following program fragment reads a character from <i>netStream</i> only if there is one waiting to be read. <pre> if Net.TokenAvailable (<i>netStream</i>) then var <i>token</i> : string get : <i>netStream</i>, <i>token</i> put <i>token</i> end if </pre>
Status	Exported qualified. This means that you can only call the function by calling Net.TokenAvailable , not by calling TokenAvailable .
See also	Net.BytesAvailable , Net.CharAvailable , and Net.LineAvailable .

Net.WaitForConnection



Syntax	Net.WaitForConnection (<i>port</i> : int , var <i>netAddr</i> : string) : int
Description	<p>Listens for a connection at the port specified by the <i>port</i> parameter. When another program connects to the port, then the function returns. The address of the connecting machine is specified in the <i>netAddr</i> parameter and the Net.WaitForConnection returns a network stream descriptor which can be used with the put, get, read, and write statements and eof function to send and receive data to the connecting program. It is also the parameter used for the Net.CloseConnection, Net.BytesAvailable, Net.CharAvailable, Net.LineAvailable, and Net.TokenAvailable functions.</p> <p>In OOT, the connection to a port is made with the Net.OpenConnection function.</p> <p>The <i>netAddr</i> parameter is a string specifying the net address of the machine that connected to the port. It is the machines numerical address.</p>

	<p>In general, system program listen in on ports with numbers below 1024. Port numbers above 1024 are generally available for use by user created programs.</p> <p>The program will wait for indefinitely for a connection to made to the port.</p>
Details	<p>The Net module requires a TCP/IP stack to be installed and operating in order to function. It does not communicate using any other protocols</p> <p>It is possible for Firewalls to interfere with the actions of the Net module, preventing connections from taking place.</p>
Example	See Net.OpenConnection for an example of Net.WaitForConnection .
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling Net.WaitForConnection, not by calling WaitForConnection.</p>
See also	Net.OpenConnection and Net.CloseConnection .

new statement

Syntax	<p>A <i>newStatement</i> is:</p> <p>new [<i>collectionOrClassId</i> ,] <i>pointerVariableReference</i></p>
Description	<p>A new statement creates (allocates) a new element and assigns its location to the pointer variable. This element can be an object of a collection or class or a value of a type. If the <i>collectionOrClassId</i> is omitted, the choice of element is determined by the type of the pointer. For example, if the pointer is to class C, an object of class C will be allocated.</p>
Example	<p>Using a collection, declare a list of records and allocate one record.</p> <pre> var list : collection of record contents : string (10) next : pointer to list % Short form: next : ^ list end record var first : pointer to list % Short form: var first : ^ list new list, first % Short form: new first </pre>
Example	<p>Using a class, create an object of that class. The object is located by the <i>start</i> pointer.</p> <pre> class node export var next, var name name : string (25) </pre>

```

    next : pointer to node % Short form: next : ^ node
end node
var start : pointer to node % Short form: var start : ^ node
    new node, start % Short form: new start

```

Example Using a record type, declare a list of records and allocate one record.

```

type item:
  record
    contents : string ( 10 )
    next : pointer to item % Short form: next : ^ item
  end record
var first : pointer to item % Short form: var first : ^ item
  new first

```

Details As the examples in this section show, a pointer can locate one of three things: an object of a collection, an object of a class or a value of a type. In the **new** statement, the *collectionOrClassId* can be omitted. If the pointer locates a type, it must be omitted. The **free** statement is used to deallocate an element.

An imported class can have one of its objects created (by the **new** statement) only if the class is imported **var**.

If there is no more space to allocate an element, **new** will set the pointer to be the **nil** value, and the program will continue executing.

If the pointer locates class *C* and *C* contains an **implement by** list, the object created by **new** is the inherited object (through any number of levels of **implement by**). The pointer, however, remains a pointer to *C*.

The form **new** *p* is a short form for **new** *C*, *p* when *C* is the class or collection given in *p*'s type.

If *p* is a pointer to class *C* and *C* has a descendant (expansion) class *D*, a **new** statement can be used to allocate an object of type *D*, as in:

```

    new D, p % Allocates an object of class D

```

If *D* has an **implement by** clause, the expansion is created.

Details The **new** statement can also be used to resize a **flexible array**. If an array has been declared flexible using the syntax .

```

var name : flexible array indexType { , indexType } of typeSpec

```

The indices may have compile-time or run-time upper bounds (the lower bound must be compile-time). The upper bounds can be changed by using:

```

    new name , newUpper1 {, newUpper2}

```

The existing array entries will retain their values, except that any index made smaller will have the corresponding array entries lost. Any index made larger will have the new array entries uninitialized (if applicable).

Additionally, the upper bound (both in the declaration and the **new** statement) can be made one less than the lower bound. This effectively makes an array that contains 0 elements. It can later be increased in size with another **new**.

In the current implementation (1999), with a multi-dimensional array with a non-zero number of total elements, it is a run-time error to change any but the first dimension (unless one of the new upper bounds is one less than the corresponding lower bound, giving 0 elements in the array) as the algorithm to rearrange the element memory locations has not yet been implemented.

Currently, only variables can be declared in this form. There is no flexible array parameter type, although a flexible array can be passed to an array parameter with "*" as the upper bound.

Example See **array** for an example of **flexible arrays**.

See also **class** and **collection** declarations, **pointer** type, **free** statement, **nil** value and **implement by** list.
For flexible arrays, see also **array** and **flexible**.

nil pointer to a collection

Syntax **nil** [(*collectionOrClassId*)]

Description The **nil** pointer does not locate any element (object). Pointers locate items in collections, classes and types. The *collectionOrClassId* is optional.
This **nil** pointer is distinct from pointers to actual elements, and it can be compared to these pointers. It is also distinct from the uninitialized pointer value.

Example In this example, the pointer called *first* is set to the nil pointer of collection *c*, that is, to **nil**(*c*).

```
var c : collection of
  record
    name : string ( 50 )
    next : pointer to c
  end record
var first : pointer to c := nil ( c )
```

Details See also collection, class and pointer. When nil is written without the *collectionOrClassId*, it can be assigned to a pointer to any collection, class or type.

The type of **nil** without the *collectionOrClassId* is effectively a pointer to *everyClass*, an imaginary class that has no objects and is the descendant of all classes. This implies that it can be assigned to any other class pointer, because it is a descendant of all classes.

Turing allows you to write **nil** (*id*) after a forward declaration of *id* (the name of a collection, class or type) before (and after) the resolution of the *id*.

not true/false (boolean) operator

Syntax	not
Description	The not (<i>boolean negation</i>) operator produces the opposite of a true/false value. For example, not ($x > y$) is equivalent to $x \leq y$.
Example	<pre>var error : boolean := false var success : boolean ... success := not error % success becomes the opposite of error</pre>
Details	<p>The not operator takes true and produces false and takes false and produces true. The not operator can be written as \sim. See also the boolean type, <i>prefix operators</i>, and <i>precedence</i> of operators.</p> <p>The not operator can be applied to sets.</p>

objectclass of a pointer

Syntax	objectclass (<i>pointerExpn</i>)
Description	The objectclass attribute is used to find the class of an object located by a pointer. The <i>pointerExpn</i> must be an expression that is a pointer to a class.

Example See **class** for an example of classes and inheritance, in which a class called *TextFile* is inherited by a class called *Device*. The *Device* class adds a new exported procedure called *ioCtl*. In the present example, **objectclass** is used to test to make sure that the *textFilePtr* currently locates an object that was created as a *Device* (or as a descendant of *Device*). The notation *Device(textFilePtr)* converts the pointer to be a pointer to a *Device* so that *ioCtl* can be called.

```
var textFilePtr : ^ TextFile
...
if objectclass ( textFilePtr ) >= Device then
    % Can safely treat object as a Device
    Device ( textFilePtr ) . ioCtl
...
end if
```

Details This example uses the class comparison operator >= which means "is a descendant of". See **class**.

You can only use **objectclass** in class comparisons. In particular, **objectclass** cannot be used to declare pointers. For example, this:

```
var p : ^objectclass (q)
```

is not allowed.

opaque type

Description When a type *T* is exported from module, monitor or class *M* using the keyword **opaque**, the type *M.T* is distinct from all other types. Opaque types are used to guarantee that updates to values of the type are done within *M*.

See also **module** declarations for an example of an opaque type used to implement complex arithmetic. See also *equivalence* of types for the definition of the type matching rules for opaque types.

open file statement

Syntax An *openStatement* is one of:

- (a) **open** : *fileNumberVar*, *fileName*,
ioCapability
- (b) **open** : *fileNumberVar*, *argNum*, *ioCapability*
{ , *ioCapability* }

Description The **open** statement connects the program to a file so the program can perform operations such as **read** on the file. In form (a), the **open** statement translates a *fileName*, such as "Master", to a file number such as 5. Form (b), which is less-commonly used, opens a file whose name is given by a program argument. This is described below.

The **read** statement uses the file number, not the file name, to access the file. When the program is finished using the file, it disconnects from the file using the **close** statement. Each *ioCapability* is the name of an operation, such as **read**, that is to be performed on the file.

Example This programs illustrates how to open, read and then close a file.

```
var fileName : string := "Master" % Name of file
var fileNo : int          % Number of file
var inputVariable : string ( 100 )
open : fileNo, fileName, read
...
read : fileNo, inputVariable
...
close : fileNo
```

Details The **open** statement always sets the *fileNumber* to a positive number. If the **open** fails (generally because the file does not exist), the *fileNumber* is set to zero.

An *ioCapability* is one of:

get, put, read, write, seek, mod

A file can be accessed using only the statements corresponding to the input/output capabilities with which it was opened. Note: **tell** is allowed only if the open is for **seek**.

The **open** statement truncates the file to length zero if the *ioCapabilities* includes **put** or **write** but not **mod** (which stands for **modify**). In all other cases, **open** leaves the existing file intact. The **mod** *ioCapability* specifies that the file is to be modified without being truncated. Each **open** positions to the beginning of a file. There is no mechanism to delete a file.

To open for appending to the end of the file, one has to **open** for **seek, mod** and for **write** or **put** and then **seek** to the end of the file. See the **seek** statement.

Mixed mode files, which combine **get** and **read** (or **put** and **write**), are supported by some operating systems, such as UNIX, but not by others, such as MS-DOS.

On the IBM PC, one should note that opening files in other directories uses the backslash character. This is because the backslash is a special character in Turing (as in `\t` for tab and `\n` for a newline). To get a single backslash, use `\\`.

e.g. **open** : f, "C:\\STUDENTS\\SMITH\\ACCT.DAT", **put**

Form (b) of the syntax allows you to open a file whose name is given as a program argument on the command line. For example, under UNIX, the command line:

`prog.x infile outfile`

specifies to execute *prog.x* with program arguments *infile* and *outfile*.

Similarly, in the Turing programming environment, the **run** command can accept program arguments. The *argNumber* is the position of the argument on the command line. (The first argument is number 1.) The name of the file to be opened is the corresponding program argument. If there is no such argument, or if the file cannot be opened successfully, *fileNumberVariable* is set to zero. See also **nargs**, which gives the number of arguments, and **fetcharg**, which gives the *n*-th argument string.

Program argument files referenced by argument number and used in **put**, **get**, **read** or **write** statements need not be explicitly opened, but are implicitly opened with the capability corresponding to the input/output statement in which they are first used. (The *fileNumber* gives the number of the argument.)

The operating system standard files (error, output and input) are accessed using file numbers 0, -1, and -2, respectively (although this may be subject to change). These files are not opened explicitly, but are used simply by using form (b) with the number. Beware of the anomalous case of a failed open that gives you file number 0. A subsequent use of this number in a **put** will produce output that goes to the standard error stream, with no warning that the file you attempted to open is not actually being used.

To append to a file, the file must be opened with the **mod** and **seek** capability and then there must be a seek to the end of file. For example:

```
var streamnumber : int
open : streamnumber, "myfile", put, mod, seek
seek : streamnumber, *
put : streamnumber, "This appears at the end of the file"
```

There is an older and still acceptable version of **open** that has this syntax:

```
open ( var fileNumber : int, fileName : string, mode : string)
```

The *mode* must be "r" (for **get**) or "w " (for **put**).

Details

The path name specified in the open statement and elsewhere can always be in UNIX format (i.e. with forward slashes, an initial forward slash indicating an absolute directory). On the PC, absolute paths would have the form:

`a:/dir1/dir2/filename`

On the Macintosh, they would have the form:

`/volume name/directory1/directory2/file name`

Note that in addition to the UNIX path format, on the PC, you can always use standard PC path notation and on the Macintosh, you can use standard Macintosh path notation. On the Macintosh volume, directory and file names can have spaces in them.

All routines (such as the File and Dir module routines) will return file names in UNIX format, regardless of the machine the program is run on.

See also **close**, **get**, **put**, **read**, **write**, **seek** and **tell** statements.

Or operator

Syntax ***A or B***

Description The **or** (boolean) operator yields a result of true if at least one (or both) of the operands is true. **or** is a short circuit operator. For example, if *A* is true in ***A or B*** then *B* is not evaluated.

Example

```
var success : boolean := false
var continuing := true % the type is boolean
...
      continuing := continuing or success
```

Details *continuing* is set to **false**, if and only if, both *continuing* and *success* are **false**. Since Turing uses short circuit operators, once *continuing* is true, *success* will not be looked at.

The **or** operator can be applied to natural numbers. The result is the natural number that is the bit-wise **or** of the operands. See **nat** (natural number).

See also **boolean** (which discusses **true/false** values), *explicitTrueFalseConstant* (which discusses the values **true** and **false**), *precedence* and *expn* (expression).

ord character-to-integer function

Syntax	ord (<i>ch</i> : char) : int
Description	The ord function accepts an enumerated value, char , or a string of length 1, and returns the position of the value in the enumeration, or of the character in the ASCII (or EBCDIC for IBM mainframes) sequence. Values of an enumerated type are numbered left to right starting at zero. For example, ord ("A") is 65. The ord function is the inverse of chr , so for any character <i>c</i> , chr (ord (<i>c</i>)) = <i>c</i> .
See also	chr , intstr and strint functions.

palette graphics procedure



Syntax	palette (<i>p</i> : int)
Description	The palette procedure is used to change the palette number to <i>p</i> .
Example	<p>This program shows all the colors of palette number 3 for an IBM PC compatible using CGA graphics. The first line of output, for color 0, will not be visible, because the background is also color 0.</p> <pre>setscreen ("graphics") palette (3) for colorNumber : 0 .. maxcolor color (colorNumber) put "Color number ", colorNumber end for</pre>
Details	<p>The palette depends on the display hardware on the computer. On IBM PC compatibles under CGA (the default graphics mode), there are palettes numbered 0 to 3. The main palettes are numbers 2 and 3. Here is the meaning of the color numbers under these CGA palette numbers.</p> <p>Palette 2: 1 = cyan (blue), 2 = magenta (red), and 3 = white.</p> <p>Palette 3: 1 = green, 2 = red, 3 = brown.</p>

Palette number 0 is like 2 but not as bright. Palette 1 is like 3 but not as bright.

The **palette** procedure is meaningful only in a "graphics" mode. See **setscreen** for a description of the graphics modes.

See also **whatpalette**, which is used to determine the current palette number. See also **drawdot** and **maxcolor**.

See also predefined unit **View**.

parallelget parallel port function



Syntax **parallelget : int**

Description The **parallelget** procedure is used on a PC to read the value of certain pins on the parallel port. This port corresponds to the MS-DOS device "LPT1". This procedure can be used to control robots and peripherals.

Example This program reads and prints the values of the five data pins of the PC's parallel port.

```
const val : int := parallelget % Read in the set of pin values
put "Pin 10 is: ", (val div 64) mod 2
put "Pin 11 is: ", (val div 128) mod 2
put "Pin 12 is: ", (val div 32) mod 2
put "Pin 13 is: ", (val div 16) mod 2
put "Pin 15 is: ", (val div 8) mod 2
```

Details The five pins that are used for parallel input are pins 10-15. The **parallelget** procedure returns the sum of

64	Pin 10 high
128	Pin 11 high
32	Pin 12 high
16	Pin 13 high
8	Pin 15 high

The **mod** and **div** operators can be used to determine which pins are high or low.

See also the **parallelput** procedure for a diagram of the pins. That procedure is used to set the values on the parallel port.

parallelput parallel port procedure

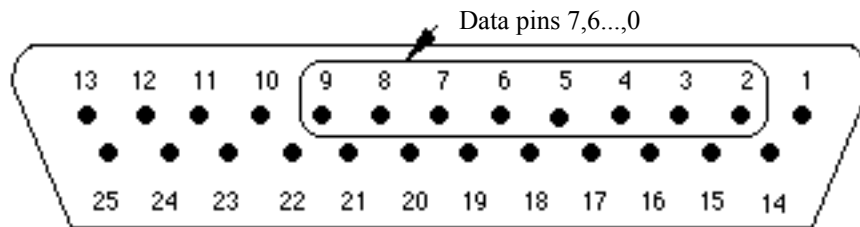
Syntax **parallelput** (*p* : int)

Description The **parallelput** procedure is used on a PC to set the values on the data pins on the parallel port. This port corresponds to the MS-DOS device "LPT1". This procedure can be used to control robots and peripherals.

Example This program sets data bit 0, data bit 1 and so on to data bit 7.

```
for i: 0..7
  parallelput (2 ** i) %Set data bit i on the parallel port
  put "Data bit ", i, " or Pin ", i + 2, "has just been set"
end for
```

Details The **parallelput** procedure is used to set the eight data bits on the PC's parallel port. These data bits 0 - 7 correspond to pins 2 - 9 on the parallel port.



Parallel Pin Out - From the IBM PC

The value sent to **parallelput** is the sum of the following:

1	Data bit 0	16	Data bit 4
2	Data bit 1	32	Data bit 5
4	Data bit 2	64	Data bit 6
8	Data bit 3	128	Data bit 7

For example, the command **parallelput** (97) sets data bits 0, 5 and 6 high (97 = 1 + 32 + 64) and sets the other data pins low. Because there are only 8 data pins in the parallel port, the value passed to **parallelput** must be from 0 to 255.

The **parallelput** procedure is not meant for sending a stream of characters to the parallel port (for example, if you want to send the string "Hello" to the printer). If you want to do this, open the file "LPT1" using the **open** statement and **put** to the file.

See also the **parallelget** function, which is used to read the values of pins on the parallel port.

paramDeclaration parameter declaration

Syntax A *paramDeclaration* is one of:

- (a) [**var**] *id* { , *id* } : *typeSpec*
- (b) *subprogramHeader*

Description A parameter declaration, which is part of the header of a procedure or function, specifies a formal parameter (see also **procedure** and **function** declarations). Form (a) above is the most common case. Form (b) specifies procedures and functions that are themselves passed as parameters.

Example

```
procedure putTitle ( title : string )
    % The parameter declaration is: title : string
    put title
end putTitle

procedure x (var s : array 1 .. * of string (*))
    % Set each element of s to the null string
    for i : 1 .. upper ( s )
        s ( i ) := ""
    end for
end x
```

Details Parameters to a procedure may be declared using **var**, which means that the parameter can be changed inside the procedure. For example, *s* is changed in the *x* procedure. If a parameter is declared without **var**, it cannot be changed. (This differs from Pascal, where non-**var** parameters can be changed.) Parameters to functions cannot be declared to be **var**. Parameters declared **var** are passed by reference, which means that a pointer to the value is passed to the procedure, rather than passing the actual value. This implies that in the call *p* (*a* (*i*)), in which array element *a*(*i*) is passed to procedure *p*, a change to *i* in *p* does not change the element referred to by *p*'s actual parameter. Every non-scalar (not integer, subrange, real, boolean, enumerated, pointer or the **char** type) parameter is passed by reference whether or not it is declared **var**. In all other cases (scalar non-**var** parameters) the parameter is passed by value (the actual value is copied to the procedure).

The upper bound of an array or string that is a formal parameter may be specified as an asterisk (*), as is done above for parameter *s* in procedure *x*. This specifies that the size of the upper bound is inherited from the corresponding actual parameter. Parameters declared using star are called *dynamic* parameters.

The names of the formal parameters must be distinct from each other, from the procedure or function name, and from pervasive identifiers. However, they need not be distinct from names outside of the procedure or function.

Example Find the zero of function *f*. This example illustrates form (b), which is a parameter that is a function. See also *subprogramHeader*.

```

function findZero ( function f ( x : real ) : real,
                    left, right, accuracy : real ) : real
  pre sign ( f ( left ) ) not= sign ( f ( right ) )
    and accuracy > 0
  var L : real := left
  var R : real := right
  var M : real
  const signLeft := sign ( f ( left ) )
  loop
    M := ( R + L ) / 2
    exit when abs ( R - L ) <= accuracy
    if signLeft = sign ( f ( M ) ) then
      L := M
    else
      R := M
    end if
  end loop
  result M
end findZero

```

Details Form (b) of *paramDeclaration* is used to specify formal parameters that are themselves procedures or functions. For example, in the *findZero* function, *f* is a formal parameter that is itself a function. The subprogram type can be used to replace form (b). In particular, the header to the *findZero* function can be replaced by the following with no change in the action. The names *g* and *x* serve no purpose, except as place holders in the declaration of *f*.

```

function findZero ( f : function g ( x : real ) : real,
                    left, right, accuracy : real ) : real

```

Details Parameters that are declared non **var** should, in principle, be constant. Unfortunately, there is an anomalous situation in which these can change. This occurs when the parameter is passed by reference, because it is a non scalar such as a string. If the actual parameter is changed while the subprogram is executing, the formal parameter will change as well. You can also optionally use the **register** keyword to request that the variable be placed in a machine register. This changes form (a) to allow the optional use of the **register** keyword. The syntax for form (a) is actually:

```

[ var ] [ register ] id { , id } : [ cheat ] typeSpec

```


In the current (1999) implementation, programs are run interpretively using pseudo-code, which has no machine registers, and the **register** keyword is ignored. See **register** for restrictions on the use of register parameters.

The optional keyword **cheat** means that the parameter has a type cheat. See **cheat**. Any variable or constant non scalar (in other words, items passed by reference) can be passed to a type cheat parameter. The internal representation will be interpreted as a value of the specified type. This is dangerous as it provides unconstrained access to the underlying computer memory.

Example This procedure outputs the values of n bytes starting at the address of formal parameter a , using a parameter type cheat.

```
procedure dump (a : cheat array 0 .. 10000 of nat1, n : int )
  for i : 0 .. n - 1
    put i, a ( i ) : 4
  end for
end dump

var s : string := "abc"
dump ( s, 5 )      % Dumps 5 bytes, starting with "abc"
```

pause statement

Syntax A *pauseStatement* is:

pause *expn*

Description The **pause** statement blocks the program (or just the process in the case of a concurrent program) for a given number of simulated time units. The *expn* must be a non-negative **int** value giving the number of time units. This is analogous to the **delay** statement, which causes blocking for a given amount of real time (actual physical time).

The interpreter maintains a counter which it considers to be simulated time. The only execution that causes this counter to increase is the **pause** statement. The process executing the **pause** is blocked until the counter has counted forward the number of units given by *expn*. All other statements (except **wait**) are considered to be infinitely fast. Several processes can be executing **pause** statements simultaneously.

The use of simulated time allows Turing to be used as a simulation language in which the **pause** statement simulates the passage of time in the simulated system.



Description	<p>This unit contains the predefined subprograms that deal with direct access to the hardware under the IBM PC architecture (available only under DOS OOT).</p> <p>All routines in the PC unit are exported qualified (and thus must be prefaced with "PC.").</p>	
Entry Points	InPort	Returns a nat1 (byte) value from a specified PC port.
	InPortWord	Returns a nat2 (word) value from a specified PC port.
	OutPort	Sends a nat1 (byte) value to a specified PC port.
	OutPortWord	Sends a nat2 (word) value to a specified PC port.
	Interrupt	Calls a PC interrupt specifying the registers to be passed to the interrupt handler.
	InterruptSegs	Calls a PC interrupt specifying the segments and registers to be passed to the interrupt handler.
	ParallelGet	Returns the value of the pins set on the parallel port.
	ParallelPut	Sets the values of the pins on the parallel port.
	SerialGet	Reads a character from the serial port.
	SerialPut	Sends a character to the serial port.
	SerialHasch	Returns whether there's a character waiting on the serial port.



Syntax **PC.InPort** (*portAddress* : nat2) : nat1

Description **PC.InPort** returns a nat1 (byte) value from a specified PC port. Ports on the IBM PC control a variety of hardware related activities such as video color maps and refresh rates, disk I/O activity, etc. Note that it's very easy to crash the machine with an incorrect *portAddress*. Read the hardware manuals to determine the correct *portAddress*.

Example The program prints out the value of the current overscan color on VGA displays (this will usually be 0) and then sets the value to colour 4.

```
% This only works on VGA systems
% Determine the address of the Attribute Controller Port
var crtStatus : int := nat2@(16#463) + 6
var prevColor : int
dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#11)      % Set the register number
prevColor := PC.InPort (16#3C1) % Read the register contents
dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#20)      % Restore the palette
put "The overscan color was ", prevColor
Input.Pause                     % Wait for a keystroke

dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#11)      % Set the register number
PC.OutPort (16#3C0, 4)          % Write the register contents
dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#20)      % Restore the palette
put "The overscan color is now 4 "
Input.Pause                     % Wait for a keystroke

dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#11)      % Set the register number
PC.OutPort (16#3C0, prevColor) % Write the register contents
dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#20)      % Restore the palette
put "The overscan color is now ", prevColor
```

Status Exported qualified.
This means that you can only call the function by calling **PC.InPort**, not by calling **InPort**.

PC.InPortWord



Syntax **PC.InPortWord** (*portAddress* : nat2) : nat2

Description	PC.InPortWord returns a nat2 (word) value from a specified PC port. Ports on the IBM PC control a variety of hardware related activities such as video color maps and refresh rates, disk I/O activity, etc. Note that it's very easy to crash the machine with an incorrect <i>portAddress</i> . Read the hardware manuals to determine the correct <i>portAddress</i> .
Status	Exported qualified. This means that you can only call the function by calling PC.InPortWord , not by calling InPortWord .

PC.Interrupt

Win DOS Mac

 X TOOT Term

Syntax	PC.Interrupt (<i>interrupt</i> : nat2, var <i>eax, ebx, ecx, edx, esi, edi, cflag</i> : nat)
Description	<p>PC.Interrupt calls the specified interrupt after loading the 8x86 registers with the values passed as parameters. After calling the interrupt, the values of the parameters are assigned from the registers.</p> <p>To determine the interrupts and register arguments, consult a DOS, BIOS or PC hardware reference manual.</p> <p>Note that it's very easy to crash the machine with an incorrect call to PC.Interrupt. Before running any program that uses the PC.Interrupt subprogram, save the program.</p>
Example	<p>In "screen" mode, foreground colors 16-31 normally cause characters to blink. This reprograms the video controller to eliminate blinking. Any character that was blinking will now have a high intensity background color. This program flips between blinking and high intensity background.</p> <pre> % This only works on EGA/VGA systems for back : 0 .. 7 for fore : 0 .. 31 Text.Locate (back + 1, fore + 1) Text.Color (fore) Text.ColorBack (back) put "X" .. end for end for end for Input.Pause % Wait for a keystroke var ax, bx, cx, dx, si, di, flag : nat % Turn off blinking, use high intensity background instead ax := 16#1003 </pre>

bx := 0

PC.Interrupt (16#10, *ax, bx, cx, dx, si, di, flag*)

Input.Pause % *Wait for a keystroke*

% *Turn on blinking, no high intensity background colors*

bx := 1

PC.Interrupt (16#10, *ax, bx, cx, dx, si, di, flag*)

Details

Under DOS OOT, the processor is running in 386 or 32 bit mode. This means that all addresses are 32 bits and must be set as such for those routines that involve passing an address to an interrupt. In general, the segment register should be set to 0 and the register should be set to the flat address of the object. The flat address is calculated by multiplying the segment by 16 and adding the offset.

The DOS extender will translate the addresses into standard 16 bit addresses and back again. While most of the available interrupts are supported by the DOS extender, check with Holt Software before trying to use an unusual interrupt.

To set various registers such as *al, ah*, etc., use the following guide:

To set *al* in *eax*, use **bits** (*eax*, 0 .. 7) := *value*

To set *ah* in *eax*, use **bits** (*eax*, 8 .. 15) := *value*

To set *ax* in *eax*, use **bits** (*eax*, 0 .. 15) := *value*

See also

PC.InterruptSegs for calling interrupts where the segment registers need to be set.

Status

Exported qualified.

This means that you can only call the function by calling **PC.Interrupt**, not by calling **Interrupt**.

PC.InterruptSegs

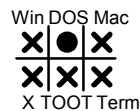


Syntax

PC.InterruptSegs (*interrupt* : **nat2**,
 var *eax, ebx, ecx, edx, esi, edi, cflag* : **nat**,
 var *ds, es, ss, cs* : **nat**)

Description	<p>PC.InterruptSegs calls the specified interrupt after loading the 8x86 registers and segment registers with the values passed as parameters. After calling the interrupt, the values of the parameters are then assigned from the registers and segment registers.</p> <p>To determine the interrupts and register arguments, consult a DOS, BIOS or PC hardware reference manual.</p> <p>Note that it's very easy to crash the machine with an incorrect call to PC.InterruptSegs. Before running any program that uses the PC.InterruptSegs subprogram, save the program.</p>
Details	<p>Under DOS OOT, the processor is running in 386 or 32 bit mode. This means that all addresses are 32 bits and must be set as such for those routines that involve passing an address to an interrupt. In general, the segment register should be set to 0 and the register should be set to the flat address of the object. The flat address is calculated by multiplying the segment by 16 and adding the offset.</p> <p>The DOS extender will translate the addresses into standard 16 bit addresses and back again. While most of the available interrupts are supported by the DOS extender, check with Holt Software before trying to use an unusual interrupt.</p> <p>To set various registers such as al, ah, etc., use the following guide:</p> <p style="padding-left: 40px;">To set al in eax, use bits (<i>eax</i>, 0 .. 7) := <i>value</i></p> <p style="padding-left: 40px;">To set ah in eax, use bits (<i>eax</i>, 8 .. 15) := <i>value</i></p> <p style="padding-left: 40px;">To set ax in eax, use bits (<i>eax</i>, 0 .. 15) := <i>value</i></p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling PC.InterruptSegs, not by calling InterruptSegs.</p>

PC.OutPort



Syntax	PC.OutPort (<i>portAddress</i> : nat2 , <i>value</i> : nat1)
Description	<p>PC.OutPort writes a nat1 (byte) value to a specified PC port. Ports on the IBM PC control a variety of hardware related activities such as video color maps and refresh rates, disk I/O activity, etc. Note that it's very easy to crash the machine with an incorrect <i>portAddress</i>. Read the hardware manuals to determine the correct <i>portAddress</i>.</p>

Example The program prints out the value of the current overscan color on VGA displays (this will usually be 0) and then sets the value to colour 4.

```
% This only works on VGA systems
% Determine the address of the Attribute Controller Port
var crtStatus : int := nat2@(16#463) + 6
var prevColor : int
dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#11)      % Set the register number
prevColor := PC.InPort (16#3C1) % Read the register contents
dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#20)      % Restore the palette
put "The overscan color was ", prevColor
Input.Pause                     % Wait for a keystroke

dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#11)      % Set the register number
PC.OutPort (16#3C0, 4)          % Write the register contents
dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#20)      % Restore the palette
put "The overscan color is now 4 "
Input.Pause                     % Wait for a keystroke

dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#11)      % Set the register number
PC.OutPort (16#3C0, prevColor)  % Write the register contents
dummy := PC.InPort (crtStatus) % Toggle the controller
PC.OutPort (16#3C0, 16#20)      % Restore the palette
put "The overscan color is now ", prevColor
```

Status Exported qualified.
This means that you can only call the function by calling **PC.OutPort**, not by calling **OutPort**.

PC.OutPortWord



Syntax **PC.OutPortWord** (*portAddress* : **nat2**, *value* : **nat2**)

Description **PC.OutPortWord** writes a nat2 (word) value to a specified PC port. Ports on the IBM PC control a variety of hardware related activities such as video color maps and refresh rates, disk I/O activity, etc. Note that it's very easy to crash the machine with an incorrect *portAddress*. Read the hardware manuals to determine the correct *portAddress*.

Status Exported qualified.
 This means that you can only call the function by calling **PC.OutPortWord**, not by calling **OutPortWord**.

PC.ParallelGet



Syntax **PC.ParallelGet** (*port* : int) : nat1

Description The **PC.ParallelGet** function is used to read the value of certain pins on a parallel port. The port is specified with the *port* parameter which can have the value 1, 2 or 3 corresponding to "LPT1", "LPT2" and "LPT3". This procedure can be used to control robots and peripherals.

Example This program reads and prints the values of the five data pins of the PC's parallel port.

```
% Read in the set of pin values from LPT1
const val : int := PC.ParallelGet (1)
put "Pin 10 is: ", (val div 64) mod 2
put "Pin 11 is: ", (val div 128) mod 2
put "Pin 12 is: ", (val div 32) mod 2
put "Pin 13 is: ", (val div 16) mod 2
put "Pin 15 is: ", (val div 8) mod 2
```

Details The five pins that are used for parallel input are pins 10-15. The **PC.ParallelGet** procedure returns the sum of

64	Pin 10 high
128	Pin 11 high
32	Pin 12 high
16	Pin 13 high
8	Pin 15 high

The **mod** and **div** operators can be used to determine which pins are high or low.

Status Exported qualified.
 This means that you can only call the function by calling **PC.ParallelGet**, not by calling **ParallelGet**.

See also **PC.ParallelPut** procedure for a diagram of the pins. That procedure is used to set the values on the parallel port.

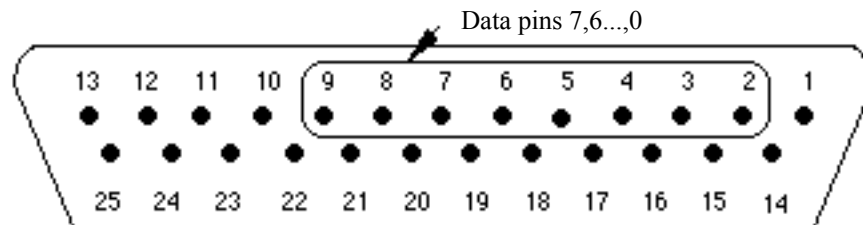
PC.ParallelPut



- Syntax** **PC.ParallelPut** (*port* : int, *value* : int)
- Description** The **PC.ParallelPut** procedure is used on a PC to set the values on the data pins on the parallel port. The port is specified with the *port* parameter which can have the value 1, 2 or 3 corresponding to "LPT1", "LPT2" and "LPT3". This procedure can be used to control robots and peripherals.
- Example** This program sets data bit 0, data bit 1 and so on to data bit 7.
- ```

for i : 0 .. 7
 %Set data bit i on parallel port LPT2
 PC.ParallelPut (2, 2 ** i)
 put "Data bit ", i, " or Pin ", i + 2, "has just been set"
end for

```
- Details** The **PC.ParallelPut** procedure is used to set the eight data bits on the PC's parallel port. These data bits 0 - 7 correspond to pins 2 - 9 on the parallel port.



Parallel Pin Out - From the IBM PC

The value sent to **PC.ParallelPut** is the sum of the following:

|   |            |     |            |
|---|------------|-----|------------|
| 1 | Data bit 0 | 16  | Data bit 4 |
| 2 | Data bit 1 | 32  | Data bit 5 |
| 4 | Data bit 2 | 64  | Data bit 6 |
| 8 | Data bit 3 | 128 | Data bit 7 |

For example, the command **PC.ParallelPut** (97) sets data bits 0, 5 and 6 high ( $97 = 1 + 32 + 64$ ) and sets the other data pins low. Because there are only 8 data pins in the parallel port, the value passed to **PC.ParallelPut** must be from 0 to 255.


The **PC.ParallelPut** procedure is not meant for sending a stream of characters to the parallel port (for example, if you want to send the string "Hello" to the printer). If you want to do this, open the file "LPT1" using the **open** statement and **put** to the file.

Status Exported qualified.

This means that you can only call the function by calling **PC.ParallelPut**, not by calling **ParallelPut**.

See also **PC.ParallelGet** function, which is used to read the values of pins on the parallel port.

## PC.SerialGet

Win DOS Mac  
  
 X TOOT Term

Syntax **PC.SerialGet** (*port* : int) : nat1

Description The **PC.SerialGet** procedure is used on a PC to read a single character from a serial port. The port is specified with the *port* parameter which can have the value 1, 2 or 3 corresponding to "COM1", "COM2" and "COM3". This function can be used to read characters from a modem or from another computer through a null-modem cable.

Example This program sends a message to the COM1 port and then gets a reply ending with a newline.

```
% We assume that the serial port has been set up previously using
% the DOS mode command.
const message : string := "This is a test"
for i : 1.. length (message)
 PC.SerialPut (1, ord (message (i)))
end for
PC.SerialPut (1, 10)
var inmessage : string := ""
var ch : int
loop
 ch := PC.SerialGet (1)
 exit when ch = 10
 inmessage := inmessage + chr (ch)
end loop

put inmessage
```

Details **PC.SerialPut** and **PC.SerialGet** assume that the port's baud rate, stop bits, etc. have been properly set with the DOS **mode** command outside of Turing. Check the DOS manual for information on how to use this command.

**PC.SerialPut** and **PC.SerialGet** will work reliably for communication up to 1200 baud. These routines are not suitable for rates higher than this.

Note that these routines pass integers that represent characters. Consequently, they should be passed (and will return) values that are between 0 and 255. Because they pass integers, use the **chr** and **ord** functions to convert integers to characters and vice-versa.

|          |                                                                                                                                                                                |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>PC.SerialGet</b> , not by calling <b>SerialGet</b> .                                       |
| See also | <b>PC.SerialPut</b> procedure which sends characters out the serial port. See also <b>chr</b> and <b>ord</b> functions for converting between a character and its ASCII value. |

## PC.SerialHasch




|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>PC.SerialHasch</b> ( <i>port</i> : int) : boolean                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | The <b>PC.SerialHasch</b> procedure is used on a PC to determine if there is a character waiting to be read on a serial port. The port is specified with the <i>port</i> parameter which can have the value 1, 2 or 3 corresponding to "COM1", "COM2" and "COM3". This function can be in conjunction with <b>PC.SerialGet</b> and <b>PC.SerialPut</b> to receive and send characters to a modem or to another computer through a null-modem cable.                  |
| Example     | <p>This program continuously send the letter "A" over COM2 until it receives the letter "B" as acknowledgment.</p> <pre> % We assume that the serial port has been set up previously using % the DOS <b>mode</b> command. loop   PC.SerialPut (2, "A")   if SerialHasch (2) then     var value : int := PC.SerialGet (2)     exit when chr (value) = "B"     put "Did not receive a \"B\" ."     put "Received a ", chr (value), "instead."   end if end loop </pre> |
| Details     | <b>PC.SerialHasch</b> assumes that the port's baud rate, stop bits, etc. have been properly set with the DOS <b>mode</b> command outside of Turing. Check the DOS manual for information on how to use this command.                                                                                                                                                                                                                                                 |

**PC.SerialHasch** will work reliably for communication up to 1200 baud. These routines are not suitable for rates higher than this.

|          |                                                                                                                                                                                                                  |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>PC.SerialHasch</b> , not by calling <b>SerialHasch</b> .                                                                     |
| See also | <b>PC.SerialGet</b> and <b>PC.SerialPut</b> procedures which receive and send characters out a serial port. See also <b>chr</b> and <b>ord</b> functions for converting between a character and its ASCII value. |

## PC.SerialPut

Win DOS Mac  
  
 X TOOT Term

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>PC.SerialPut</b> ( <i>port</i> : <b>int</b> , <i>value</i> : <b>int</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | The <b>PC.SerialPut</b> procedure is used on a PC to send a single character out the serial port. The port is specified with the <i>port</i> parameter which can have the value 1, 2 or 3 corresponding to "COM1", "COM2" and "COM3". This function can be used to send characters to a modem or to another computer through a null-modem cable.                                                                                                                                                                                                                                                                                                             |
| Example     | <p>This program sends a message to the COM2 port and then gets a reply ending with a newline.</p> <pre> % We assume that the serial port has been set up previously using % the DOS <b>mode</b> command. <b>const</b> message : <b>string</b> := "This is a test" <b>for</b> i : 1.. <b>length</b> (message)     <b>PC.SerialPut</b> (1, <b>ord</b> (message (i))) <b>end for</b> <b>PC.SerialPut</b> (1, 10) <b>var</b> inmessage : <b>string</b> := "" <b>var</b> ch : <b>int</b> <b>loop</b>     ch := <b>PC.SerialGet</b> (1)     <b>exit when</b> ch = 10     inmessage := inmessage + <b>chr</b> (ch) <b>end loop</b>      <b>put</b> inmessage </pre> |
| Details     | <b>PC.SerialPut</b> and <b>PC.SerialGet</b> assume that the port's baud rate, stop bits, etc. have been properly set with the DOS <b>mode</b> command outside of Turing. Check the DOS manual for information on how to use this command.                                                                                                                                                                                                                                                                                                                                                                                                                    |

**PC.SerialPut** and **PC.SerialGet** will work reliably for communication up to 1200 baud. These routines are not suitable for rates higher than this.

Note that these routines pass integers that represent characters. Consequently, they should be passed (and will return) values that are between 0 and 255. Because they pass integers, use the **chr** and **ord** functions to convert integers to characters and vice-versa.

|          |                                                                                                                                                                                |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>PC.SerialPut</b> , not by calling <b>SerialPut</b> .                                       |
| See also | <b>PC.SerialGet</b> procedure which sends characters out the serial port. See also <b>chr</b> and <b>ord</b> functions for converting between a character and its ASCII value. |

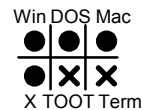
## pervasive declaration modifier

|             |                                                                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | When a variable, constant, type or subprogram is declared, you can specify that it is to be <b>pervasive</b> , which means that it does not need to be explicitly imported into modules, monitors or classes in the current scope. The keyword <b>pervasive</b> can be abbreviated as an asterisk (*). |
| Example     | <pre>var pervasive counter : int % Short form: var * count : int const * maxCounter : int := 100 procedure * p ( x : real ) ... end p</pre>                                                                                                                                                            |
| Details     | The keyword <b>pervasive</b> is also used in <b>export</b> lists along with the keyword <b>unqualified</b> . See <b>export</b> list for details.                                                                                                                                                       |
| See also    | <b>var</b> declaration, <b>const</b> declaration, <b>procedure</b> declaration, <b>function</b> declaration, <b>subprogram</b> header and <b>export</b> list for uses of <b>pervasive</b> .                                                                                                            |

# Pic

|              |                                                                                                                                                                                                                                                                                                |                                                                         |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that deal with taking pictures of part of the screen, displaying them and moving pictures from file to screen and back.</p> <p>All routines in the <b>Pic</b> unit are exported qualified (and thus must be prefaced with "<b>Pic.</b>").</p> |                                                                         |
| Entry Points | <b>New</b>                                                                                                                                                                                                                                                                                     | Creates a picture from a specified portion of the screen.               |
|              | <b>Draw</b>                                                                                                                                                                                                                                                                                    | Draws a picture on the screen.                                          |
|              | <b>Free</b>                                                                                                                                                                                                                                                                                    | Frees up the picture created by using <b>New</b> or <b>FileNew</b> .    |
|              | <b>FileNew</b>                                                                                                                                                                                                                                                                                 | Creates a picture from a file in an allowed format.                     |
|              | <b>Save</b>                                                                                                                                                                                                                                                                                    | Saves a picture as a file in an allowed format.                         |
|              | <b>ScreenLoad</b>                                                                                                                                                                                                                                                                              | Displays a file in an allowed format on the screen directly.            |
|              | <b>ScreenSave</b>                                                                                                                                                                                                                                                                              | Saves a specified portion of the screen as a file in an allowed format. |

## Pic.Draw



|             |                                                                                                                                                                                                            |                                                                                                                                                                                             |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Pic.Draw</b> ( <i>picID</i> , <i>x</i> , <i>y</i> , <i>mode</i> : <b>int</b> )                                                                                                                          |                                                                                                                                                                                             |
| Description | <p><b>Pic.Draw</b> is used to draw a picture on the screen. The picture is drawn with the lower left corner at (<i>x</i>, <i>y</i>).</p> <p>The <i>mode</i> parameter has one of the following values:</p> |                                                                                                                                                                                             |
|             | <i>picCopy</i>                                                                                                                                                                                             | This draws the picture on top of what was underneath, obscuring it completely.                                                                                                              |
|             | <i>picXor</i>                                                                                                                                                                                              | This draws the picture XORing it with the background. In DOS, you can use this function to do animation. Drawing an object on top of itself with XOR erases it and restores the background. |

|         |                      |                                                                                                                                                                                                                          |
|---------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|         | <i>picMerge</i>      | This draws the picture like <i>picCopy</i> except that any occurrence of the background color in the picture is not drawn to the screen. This allows you to draw an irregularly-shaped object and draw it to the screen. |
|         | <i>picUnderMerge</i> | This draws the picture, but only where the background color was displayed underneath it. The effect of this is to make the picture appear to be displayed behind the background.                                         |
| Details |                      | If the <b>Pic.Draw</b> call fails, <b>Error.Last</b> will return a non-zero value indicating the reason for the failure. <b>Error.LastMsg</b> will return a string which contains the textual version of the error.      |
| Details |                      | At the time of writing, under DOS OOT and X OOT, only <i>picCopy</i> and <i>picXor</i> are supported.                                                                                                                    |
| Example |                      | The program draws a graphic on the screen and then repeats it 50 times in random positions.                                                                                                                              |

```

var picID: int
var x, y : int
Draw.FillBox (50, 50, 150, 150, red)
Draw.FillStar (50, 50, 150, 150, green)
Draw.FillOval (100, 100, 50, 50, blue)

picID := Pic.New (50, 50, 150, 150)
for i : 1 .. 50
 x := Rand.Int (0, maxx) % Random x
 y := Rand.Int (0, maxy) % Random y
 Pic.Draw (picID, x, y, picCopy)
end for
Pic.Free (picID)

```

|        |                                                                                                                                 |
|--------|---------------------------------------------------------------------------------------------------------------------------------|
| Status | Exported qualified.<br>This means that you can only call the function by calling <b>Pic.Draw</b> , not by calling <b>Draw</b> . |
|--------|---------------------------------------------------------------------------------------------------------------------------------|

## Pic.FileNew



|        |                                                                     |
|--------|---------------------------------------------------------------------|
| Syntax | <b>Pic.FileNew</b> ( <i>fileName</i> : <b>string</b> ) : <b>int</b> |
|--------|---------------------------------------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |           |                                  |           |                                  |           |                                  |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|----------------------------------|-----------|----------------------------------|-----------|----------------------------------|
| Description | <p><b>Pic.FileNew</b> is used to obtain a picture from a file. The <b>Pic.FileNew</b> procedure allocates the memory for the picture, which can be very large for pictures of large areas. The memory is freed up when the program calls <b>Pic.Free</b> with the picture ID. The picture can be used with the <b>Pic.Draw</b> and <b>Pic.Save</b>.</p> <p>Various versions of OOT can convert different formats of files. Look at the release notes to see which file formats can be read and written.</p> <p>The <i>fileName</i> parameter must give the format of the file:</p> <table> <tr> <td>PCX files</td><td>"PCX:filename" or "filename.PCX"</td></tr> <tr> <td>BMP files</td><td>"BMP:filename" or "filename.BMP"</td></tr> <tr> <td>TIM files</td><td>"TIM:filename" or "filename.TIM"</td></tr> </table> | PCX files | "PCX:filename" or "filename.PCX" | BMP files | "BMP:filename" or "filename.BMP" | TIM files | "TIM:filename" or "filename.TIM" |
| PCX files   | "PCX:filename" or "filename.PCX"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |           |                                  |           |                                  |           |                                  |
| BMP files   | "BMP:filename" or "filename.BMP"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |           |                                  |           |                                  |           |                                  |
| TIM files   | "TIM:filename" or "filename.TIM"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |           |                                  |           |                                  |           |                                  |
| Details     | <p>On a DOS system, the <b>Pic.FileNew</b> routine may end up remapping the color palette.</p> <p>If the <b>Pic.FileNew</b> call fails, then it returns 0. Also <b>Error.Last</b> will return a non-zero value indicating the reason for the failure. <b>Error.LastMsg</b> will return a string which contains the textual version of the error.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |           |                                  |           |                                  |           |                                  |
| Details     | <p>At the time of this writing, DOS OOT supported only DOS OOT native graphics files (i.e those files with no recognized suffix or prefix). These files can only be used by DOS OOT. PCX support will appear in the next major release. Consult the release notes to find out which file formats are currently supported.</p> <p>At the time of writing, WinOOT supported only BMP files. Consult the release notes to find out which file formats are currently supported.</p>                                                                                                                                                                                                                                                                                                                                       |           |                                  |           |                                  |           |                                  |
| Example     | <p>The program reads a graphic from the file <i>mypic.pcx</i> and then draws it 50 times.</p> <pre> var picID: int var x, y : int  picID := Pic.FileNew ("mypic.pcx") for i : 1 .. 50   x := Rand.Int (0, maxx)      % Random x   y := Rand.Int (0, maxy)      % Random y   Pic.Draw (picID, x, y, picCopy) end for Pic.Free (picID) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |           |                                  |           |                                  |           |                                  |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Pic.FileNew</b>, not by calling <b>FileNew</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |           |                                  |           |                                  |           |                                  |

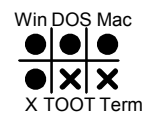


# Pic.Free



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Pic.Free</b> ( <i>picID</i> : int)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Description | <b>Pic.Free</b> is used to release the memory allocated by <b>Pic.New</b> . It frees up the memory allocated to the parameter <i>picID</i> . This means that <i>picID</i> can not be used in a <b>Draw.Pic</b> or <b>Draw.Save</b> procedure after <b>Pic.Free</b> is called.                                                                                                                                                                                                                                               |
| Details     | If <b>Pic.Free</b> is passed an invalid picture ID, a fatal error occurs. If the <b>Pic.Free</b> call fails for other (non-fatal) reasons, <b>Error.Last</b> will return a non-zero value indicating the reason for the failure. <b>Error.LastMsg</b> will return a string which contains the textual version of the error.                                                                                                                                                                                                 |
| Example     | <p>The program draws a graphic on the screen and then draws it 50 times.</p> <pre> var <i>picID</i>: int var <i>x, y</i> : int Draw.FillBox (50, 50, 150, 150, red) Draw.FillStar (50, 50, 150, 150, green) Draw.FillOval (100, 100, 50, 50, blue)  <i>picID</i> := Pic.New (50, 50, 150, 150) for i : 1 .. 50     <i>x</i> := Rand.Int (0, maxx)      % Random x     <i>y</i> := Rand.Int (0, maxy)      % Random y     Pic.Draw (<i>picID</i>, <i>x</i>, <i>y</i>, <i>picCopy</i>) end for Pic.Free (<i>picID</i>) </pre> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Pic.Free</b>, not by calling <b>Free</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                    |

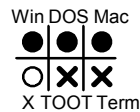
# Pic.New



|        |                                                     |
|--------|-----------------------------------------------------|
| Syntax | <b>Pic.New</b> ( <i>x1, y1, x2, y2</i> : int) : int |
|--------|-----------------------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | <p><b>Pic.New</b> is used to obtain a picture of a portion of the screen. The <b>Pic.New</b> procedure allocates the memory for the picture, which can be very large for pictures of large areas. The memory is freed up when the program calls <b>Pic.Free</b> with the picture ID. The picture can be used with the <b>Pic.Draw</b> and <b>Pic.Save</b>.</p> <p>The picture is of the screen area defined by the rectangle <math>(x1, y1) - (x2, y2)</math>.</p> |
| Details     | <p>If the <b>Pic.New</b> call fails, then it returns 0. Also <b>Error.Last</b> will return a non-zero value indicating the reason for the failure. <b>Error.LastMsg</b> will return a string which contains the textual version of the error.</p>                                                                                                                                                                                                                  |
| Example     | <p>The program draws a graphic on the screen and then draws it 50 times.</p> <pre> var picID: int var x, y : int Draw.FillBox (50, 50, 150, 150, red) Draw.FillStar (50, 50, 150, 150, green) Draw.FillOval (100, 100, 50, 50, blue)  picID := Pic.New (50, 50, 150, 150) for i : 1 .. 50     x := Rand.Int (0, maxx)      % Random x     y := Rand.Int (0, maxy)      % Random y     Pic.Draw (picID, x, y, picCopy) end for Pic.Free (picID) </pre>              |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Pic.New</b>, not by calling <b>New</b>.</p>                                                                                                                                                                                                                                                                                                                             |

## Pic.Save



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |           |                                  |           |                                  |           |                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|----------------------------------|-----------|----------------------------------|-----------|----------------------------------|
| Syntax      | <b>Pic.Save</b> ( <i>picID</i> : int, <i>fileName</i> : string)                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |           |                                  |           |                                  |           |                                  |
| Description | <p><b>Pic.Save</b> is used to save a picture on the screen to a file.</p> <p>Various versions of OOT can save different formats of files. Look at the release notes to see which file formats can be read and written.</p> <p>The <i>fileName</i> parameter must give the format of the file:</p> <table> <tr> <td>PCX files</td><td>"PCX:filename" or "filename.PCX"</td></tr> <tr> <td>BMP files</td><td>"BMP:filename" or "filename.BMP"</td></tr> <tr> <td>TIM files</td><td>"TIM:filename" or "filename.TIM"</td></tr> </table> | PCX files | "PCX:filename" or "filename.PCX" | BMP files | "BMP:filename" or "filename.BMP" | TIM files | "TIM:filename" or "filename.TIM" |
| PCX files   | "PCX:filename" or "filename.PCX"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |           |                                  |           |                                  |           |                                  |
| BMP files   | "BMP:filename" or "filename.BMP"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |           |                                  |           |                                  |           |                                  |
| TIM files   | "TIM:filename" or "filename.TIM"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |           |                                  |           |                                  |           |                                  |

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details | <p>At the time of this writing, DOS OOT supported only DOS OOT native graphics files (i.e those files with no recognized suffix or prefix). These files can only be used by DOS OOT. PCX support will appear in the next major release. Consult the release notes to find out which file formats are currently supported.</p> <p>At the time of writing, WinOOT supported only BMP files. Consult the release notes to find out which file formats are currently supported.</p>                                                                                                                                                                                            |
| Details | <p>If <b>Pic.Save</b> is passed an invalid picture ID, a fatal error occurs. If the <b>Pic.Save</b> call fails for other (non-fatal) reasons, <b>Error.Last</b> will return a non-zero value indicating the reason for the failure. <b>Error.LastMsg</b> will return a string which contains the textual version of the error.</p>                                                                                                                                                                                                                                                                                                                                         |
| Example | <p>The program draws a graphic on the screen and then saves it as a BMP file.</p> <pre> var picID: int var x, y : int Draw.FillBox (50, 50, 150, 150, red) Draw.FillStar (50, 50, 150, 150, green) Draw.FillOval (100, 100, 50, 50, blue)  picID := Pic.New (50, 50, 150, 150) Pic.Save (picID, "BMP:mypic.dat") Pic.Free (picID) </pre>                                                                                                                                                                                                                                                                                                                                   |
| Example | <p>The following two programs save and load a file in DOS OOT native format.</p> <pre> % Program to save a picture in mypic.pic var picID: int var x, y : int Draw.FillBox (50, 50, 150, 150, red) Draw.FillStar (50, 50, 150, 150, green) Draw.FillOval (100, 100, 50, 50, blue)  picID := Pic.New (50, 50, 150, 150) Pic.Save (picID, "mypic.pic") Pic.Free (picID)  % Program to load the picture back again and draw 50 copies var picID: int var x, y : int  picID := Pic.FileNew ("mypic.pic") for i : 1 .. 50   x := Rand.Int (0, maxx)      % Random x   y := Rand.Int (0, maxy)      % Random y   Pic.Draw (picID, x, y, picCopy) end for Pic.Free (picID) </pre> |
| Status  | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Pic.Save</b>, not by calling <b>Save</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

# Pic.ScreenLoad



|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|----------------------------------|-----------|----------------------------------|-----------|----------------------------------|----------------|--------------------------------------------------------------------------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax               | <b>Pic.ScreenLoad</b> ( <i>fileName</i> : <b>string</b> , <i>x</i> , <i>y</i> , <i>mode</i> : <b>int</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
| Description          | <p><b>Pic.ScreenLoad</b> displays a picture from a file straight to the screen.</p> <p>Various versions of OOT can convert different formats of files. Look at the release notes to see which file formats can be read and written.</p> <p>The <i>fileName</i> parameter must give the format of the file:</p> <table><tr><td>PCX files</td><td>"PCX:filename" or "filename.PCX"</td></tr><tr><td>BMP files</td><td>"BMP:filename" or "filename.BMP"</td></tr><tr><td>TIM files</td><td>"TIM:filename" or "filename.TIM"</td></tr></table> <p>The <i>x</i> and <i>y</i> parameters set the lower left hand corner of the picture.</p> <p>The <i>mode</i> parameter has one of the following values:</p> <table><tr><td><i>picCopy</i></td><td>This draws the picture on top of what was underneath, obscuring it completely.</td></tr><tr><td><i>picXOR</i></td><td>This draws the picture XORing it with the background. In DOS, you can use this function to do animation. Drawing an object on top of itself with XOR erases it and restores the background.</td></tr><tr><td><i>picMerge</i></td><td>This draws the picture like <i>picCopy</i> except that any occurrence of the background color in the picture is not drawn to the screen. This allows you to draw an irregularly-shaped object and draw it to the screen.</td></tr><tr><td><i>picUnderMerge</i></td><td>This draws the picture, but only where the background color was displayed underneath it. The effect of this is to make the picture appear to be displayed behind the background.</td></tr></table> | PCX files | "PCX:filename" or "filename.PCX" | BMP files | "BMP:filename" or "filename.BMP" | TIM files | "TIM:filename" or "filename.TIM" | <i>picCopy</i> | This draws the picture on top of what was underneath, obscuring it completely. | <i>picXOR</i> | This draws the picture XORing it with the background. In DOS, you can use this function to do animation. Drawing an object on top of itself with XOR erases it and restores the background. | <i>picMerge</i> | This draws the picture like <i>picCopy</i> except that any occurrence of the background color in the picture is not drawn to the screen. This allows you to draw an irregularly-shaped object and draw it to the screen. | <i>picUnderMerge</i> | This draws the picture, but only where the background color was displayed underneath it. The effect of this is to make the picture appear to be displayed behind the background. |
| PCX files            | "PCX:filename" or "filename.PCX"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
| BMP files            | "BMP:filename" or "filename.BMP"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
| TIM files            | "TIM:filename" or "filename.TIM"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
| <i>picCopy</i>       | This draws the picture on top of what was underneath, obscuring it completely.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
| <i>picXOR</i>        | This draws the picture XORing it with the background. In DOS, you can use this function to do animation. Drawing an object on top of itself with XOR erases it and restores the background.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
| <i>picMerge</i>      | This draws the picture like <i>picCopy</i> except that any occurrence of the background color in the picture is not drawn to the screen. This allows you to draw an irregularly-shaped object and draw it to the screen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
| <i>picUnderMerge</i> | This draws the picture, but only where the background color was displayed underneath it. The effect of this is to make the picture appear to be displayed behind the background.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |
| Details              | <p>On a DOS system, the <b>Pic.ScreenLoad</b> routine may end up remapping the color palette.</p> <p>If the <b>Pic.ScreenLoad</b> fails, then <b>Error.Last</b> will return a non-zero value indicating the reason for the failure. <b>Error.LastMsg</b> will return a string which contains the textual version of the error.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |           |                                  |           |                                  |           |                                  |                |                                                                                |               |                                                                                                                                                                                             |                 |                                                                                                                                                                                                                          |                      |                                                                                                                                                                                  |

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details | <p>At the time of writing, under DOS OOT and X OOT, only <i>picCopy</i> and <i>picXOR</i> are supported. As well, DOS OOT supports only "TM2" files (a file format used under Turing) and DOS OOT native graphics files (i.e those files with no recognized suffix or prefix). DOS OOT native graphics files can only be used by DOS OOT. PCX support will appear in the next major release. Consult the release notes to find out which file formats are currently supported.</p> <p>At the time of writing, WinOOT supported only BMP files. Consult the release notes to find out which file formats are currently supported.</p> <p>At the time of writing, MacOOT supported only PICT files. Consult the release notes to find out which file formats are currently supported.</p> |
| Example | <p>The program displays a picture on the screen from the PCX file <i>mypic.PCX</i>.</p> <p><b>Pic.ScreenLoad</b> ("mypic.pcx", 0, 0, <i>picCopy</i>)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Status  | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Pic.ScreenLoad</b>, not by calling <b>ScreenLoad</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## Pic.ScreenSave



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |           |                                  |           |                                  |           |                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|----------------------------------|-----------|----------------------------------|-----------|----------------------------------|
| Syntax      | <b>Pic.ScreenSave</b> ( <i>x1</i> , <i>y1</i> , <i>x2</i> , <i>y2</i> : <b>int</b> , <i>fileName</i> : <b>string</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |                                  |           |                                  |           |                                  |
| Description | <p><b>Pic.ScreenSave</b> saves a portion of the screen into a file in a format specified by the file name.</p> <p>The picture saved to the file is the portion of the screen defined by the rectangle (<i>x1</i>, <i>y1</i>) - (<i>x2</i>, <i>y2</i>).</p> <p>Various versions of OOT can convert different formats of files. Look at the release notes to see which file formats can be read and written.</p> <p>The <i>fileName</i> parameter must give the format of the file:</p> <table> <tr> <td>PCX files</td><td>"PCX:filename" or "filename.PCX"</td></tr> <tr> <td>BMP files</td><td>"BMP:filename" or "filename.BMP"</td></tr> <tr> <td>TIM files</td><td>"TIM:filename" or "filename.TIM"</td></tr> </table> | PCX files | "PCX:filename" or "filename.PCX" | BMP files | "BMP:filename" or "filename.BMP" | TIM files | "TIM:filename" or "filename.TIM" |
| PCX files   | "PCX:filename" or "filename.PCX"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |           |                                  |           |                                  |           |                                  |
| BMP files   | "BMP:filename" or "filename.BMP"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |           |                                  |           |                                  |           |                                  |
| TIM files   | "TIM:filename" or "filename.TIM"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |           |                                  |           |                                  |           |                                  |
| Details     | <p>At the time of this writing, DOS OOT supported only DOS OOT native graphics files (i.e those files with no recognized suffix or prefix). These files can only be used by DOS OOT. PCX support will appear in the next major release. Consult the release notes to find out which file formats are currently supported.</p>                                                                                                                                                                                                                                                                                                                                                                                            |           |                                  |           |                                  |           |                                  |

At the time of writing, WinOOT supported only BMP files. Consult the release notes to find out which file formats are currently supported.

**Details** If the **Pic.ScreenSave** fails, then **Error.Last** will return a non-zero value indicating the reason for the failure. **Error.LastMsg** will return a string which contains the textual version of the error.

**Example** The program draws a graphic and saves it as a PICT file called *draw*.

```
Draw.FillBox (50, 50, 150, 150, red)
Draw.FillStar (50, 50, 150, 150, green)
Draw.FillOval (100, 100, 50, 50, blue)
```

```
picID := Pic.ScreenSave (50, 50, 150, 150, "PICT:draw")
```

**Status** Exported qualified.  
This means that you can only call the function by calling **Pic.ScreenSave**, not by calling **ScreenSave**.

**play**    procedure

|             |     |     |
|-------------|-----|-----|
| Win         | DOS | Mac |
| ●           | ●   | ●   |
| ○           | ✕   | ✕   |
| X TOOT Term |     |     |

**Syntax**    **play** ( *music* : **string** )

**Description** The **play** procedure is used to sound musical notes on the computer.

**Example** This program sounds the first three notes of the C scale.  
**play** ( "cde" )

**Example** This program plays from middle C to one octave above middle C and down again in 8th notes.  
**play** ( "8cdefgab>c" )  
          **play** ( "<bagfedc" )

**Details** The **play** procedure takes strings containing characters that specify notes, rests, sharps, flats and duration. The notes are the letters a to g (or A to G). A rest is p (for pause). A sharp is + and a flat is -. The durations are 1 (whole note), 2 (half note), 4 (quarter note), 8 (eighth note) and 6 (sixteenth note). The character > raises to the next octave and < lowers. For example, this is the way to play C and then C sharp one octave above middle C with a rest between them, all in sixteenth notes: **play**(">6cpc+"). Blanks can be used for readability and are ignored by **play**.

Under some systems such as UNIX, the **play** procedure may have no effect. The current (1999) implementation does not support play.

See also the **playdone** function, which is used to see if a note has finished sounding. See also the **sound** procedure, which makes a sound of a given frequency (Hertz) and duration (milliseconds). See also predefined unit **Music**.

## playdone function



**Syntax**      **playdone : boolean**

**Description**      The **playdone** function is used to determine when notes played by the **play** procedure have finished sounding.

**Example**      This program sounds the first three notes of the C scale and outputs "All done" as soon as they are finished. Without the loop, the message would come out before the notes are finished.

```
play ("cde")
loop
 exit when playdone
end loop
put "All done"
```

**Details**      Under some systems such as UNIX, the **playdone** procedure may be meaningless.

**See also**      the **play** procedure. See also the **sound** procedure which makes a sound of a given frequency (Hertz) and duration (milliseconds). See also predefined unit **Music**.

## pointer type

**Syntax**      A *pointerType* is one of:

- (a)    **[unchecked] pointer to *collectionId***  
          % Short form: ^ *collectionId*
- (b)    **[unchecked] pointer to *classId***

*% Short form: ^ classId*  
**(c) [unchecked] pointer to typeSpec**  
*% Short form: ^ typeSpec*

**Description**      A variable declared as a pointer type is used to locate an element of a collection or class or a value of a type. The **new** statement creates a new element (or value) and places the element's location in a pointer variable. The **free** statement destroys an element located by a pointer variable.

**Example**            Using a collection, declare a list or records and allocate one record.

```

var list : collection of
 record
 contents : string (10)
 next : pointer to list
 end record
var first : pointer to list
 new list, first

```

**Example**            Create a collection that will represent a binary tree.

```

var tree : collection of
 record
 name : string (10)
 left, right : pointer to tree
 end record

var root : pointer to tree
new tree, root
 tree (root).name := "Adam"

```

**Example**            Using a class, create an object of that class. The object is located by the *start* pointer. The *name* field of the object is set to *Ed*.

```

class node
 export var next, var name
 name : string (25)
 next : pointer to node % Short form: next : ^ node
end node
var start : pointer to node % Short form: var start : ^ node
new node, start % Short form: new start
 node (start) . name := "Ed" % Short form: start->name:="Ed"

```

**Details**            For collections and classes, a pointer is effectively a subscript (an index) for that collection or class. Pointers can be assigned, compared for equality and passed as parameters.

The keywords **pointer to** can be replaced by the short form **^**, as in

**var** first : **^** item

Given a pointer *p* that locates an object of class or collection *C*, the object is referenced as *C(p)* or as the short form **^** *p*. A field *f* of the object is referenced as *C(p).f* or **^***p.f* or as the short form *p->f*. For example, in the class given above, the *name* field of the object located by the *start* pointer can be set to *Alice* by:



```
start -> name := "Alice"
```

Pointers to types use the same notation, except that pointers to types are not allowed to use the form *typeSpec(p)*. See **class** for an example of the use of a class with pointers.

The carat  $\wedge$  is called the *dereferencing operator* and has the highest precedence. For example, in  $\wedge p.a$ , the carat applies to  $p$  and not to  $p.a$ . To apply  $\wedge$  to all of  $p.a$ , use parentheses:  $\wedge(p.a)$ . Several carats can appear in a row, for example,

```
var r : $\wedge \wedge$ int
```

declares a pointer to a pointer to an integer and  $\wedge \wedge r$  is the notation for referencing the integer.

A reference cannot begin with a left parenthesis, but can be surrounded by  $\wedge(\dots)$ , as in  $\wedge(q.b)$ . If  $f$  is a parameterless function declared without parentheses that returns a pointer, the form  $\wedge f$  calls  $f$  before dereferencing the pointer.

By default, all pointers are checked. This means there is a run time test to make sure that references such as  $C(p)$  actually locate elements, i.e., that  $p$  is initialized, is not **nil** and is not *dangling* (locating an object that has been freed). This checking requires extra space (the implementation attaches a *time stamp* to each pointer and object) and time. In high-performance programs in which this extra space and time are not acceptable, the pointer can be declared to be **unchecked**. When this is done, the program becomes *dangerous* and it is the programmer's responsibility to make sure that all pointer usage is valid.

If this is not the case, the program becomes susceptible to uncontrolled crashes.

Checked pointers cannot be assigned to unchecked pointers nor vice versa. However, you may, at your peril, use an implementation-dependent *type cheat*, to convert a checked pointer to a unchecked pointer, as in:

```
type checkedPtr : \wedge R
type uncheckedPtr : unchecked \wedge R
var c : checkedPtr % c is an checked pointer
var u : uncheckedPtr % u is an unchecked pointer
...
u := cheat (uncheckedPtr, d) % This is a type cheat
```

Unchecked pointers are equivalent to the pointers of the C language, which are inherently error prone and cause difficult to locate bugs. An entire collection (but not a class) can be declared unchecked, in which case all of its pointers are implicitly unchecked. See **collection**.

See also **inherit** lists for a description of the assignability rules for pointers. See **classes** and **collections** for more details about the use of pointers. See also **new** and **free** statements. See also **nil**, **objectclass** and **anyclass**.

## post assertion

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | An <i>postAssertion</i> is:<br><b>post</b> <i>trueFalseExpn</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Description | A <b>post</b> assertion is a special form of an <b>assert</b> statement that is used in a procedure or function. It is used to give requirements that the body of the procedure or function is supposed to satisfy. These requirements are given by the <i>trueFalseExpn</i> . After the body has executed and just before the procedure or function returns, the <i>trueFalseExpn</i> is evaluated. If it is true, all is well and execution continues. If it is false, execution is terminated with an appropriate message. See <b>assert</b> statements and <b>procedure</b> and <b>function</b> declarations for more details. See also <b>pre</b> and <b>invariant</b> assertions. |
| Example     | <p>This function is supposed to produce an integer approximation of the square root of integer <i>i</i>. The post condition requires that this result, which is called <i>answer</i>, must be within a distance of 1 from the corresponding <b>real</b> number square root.</p> <pre><b>function</b> <i>intSqrt</i> ( <i>i</i> : <b>int</b>) <i>answer</i> : <b>int</b>   <b>pre</b> <i>i</i> &gt;= 0   <b>post</b> <i>abs</i> ( <i>answer</i> - <i>sqrt</i> ( <i>i</i> ) ) &lt;= 1   ... <i>statements to approximate square root</i> ... <b>end</b> <i>intSqrt</i></pre>                                                                                                              |
| Details     | A <b>post</b> assertion can also be used in a module, monitor, class or process declaration to make sure that the initialization satisfies its requirements.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| See also    | <b>module</b> and <b>process</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

## pre assertion

|        |                                                               |
|--------|---------------------------------------------------------------|
| Syntax | An <i>preAssertion</i> is:<br><b>pre</b> <i>trueFalseExpn</i> |
|--------|---------------------------------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | A <b>pre</b> assertion is a special form of an <b>assert</b> statement that is used at the beginning of a procedure or function. It is used to give requirements that the caller of the procedure or functions is supposed to satisfy. These requirements are given by the <i>trueFalseExpn</i> . The <i>trueFalseExpn</i> is evaluated. If it is true, all is well and execution continues. If it is false, execution is terminated with an appropriate message. See <b>assert</b> statements and <b>procedure</b> and <b>function</b> declarations for more details. See also <b>post</b> and <b>invariant</b> assertions. |
| Example     | <p>This function computes the average of <i>n</i> values. Its <b>pre</b> condition requires that <i>n</i> must be strictly positive, to avoid the possibility of dividing by zero when computing the average.</p> <pre> <b>function</b> <i>average</i> ( <i>a</i> : <b>array</b> 1 .. * <b>of</b> <i>real</i>, <i>n</i> : <i>int</i> ) : <i>real</i>   <b>pre</b> <i>n</i> &gt; 0   <b>var</b> <i>sum</i> : <i>real</i> := 0   <b>for</b> <i>i</i> : 1 .. <i>n</i>     <i>sum</i> := <i>sum</i> + <i>a</i> ( <i>i</i> )   <b>end for</b>   <b>result</b> <i>sum</i> / <i>n</i> <b>end</b> <i>average</i> </pre>              |
| Details     | A <b>pre</b> assertion can also be used in a <b>module</b> , <b>monitor</b> , <b>class</b> or <b>process</b> declaration to make sure that requirements for initialization are met.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| See also    | <b>module</b> and <b>process</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## precedence of operators

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | <p>Turing's <i>precedence</i> rules determine the order of applying operators in an expression such as <math>3 + 4 * 5</math>. These rules state, for example, that multiplication is done before addition, so this expression is equivalent to <math>3 + (4 * 5)</math>.</p> <p>Parenthesized parts of an expression are evaluated before being used. For example, in <math>(1 + 2) * 3</math>, the addition is done before the multiplication.</p> <p>The precedence rules are defined by this table, in which operators appearing earlier in the table are applied first. For example, multiplication is applied before addition:</p> <ol style="list-style-type: none"> <li>(1)     <b>**</b>, <b>^</b>, <b>#</b></li> <li>(2)     <i>prefix</i> + and -</li> <li>(3)     <b>*</b>, <b>/</b>, <b>div</b>, <b>mod</b>, <b>rem</b>, <b>shr</b>, <b>shl</b></li> <li>(4)     <i>infix</i> +, -, <b>xor</b></li> </ol> |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- (5) <, >, =, <=, >=, **not=**, **in**, **not in**
- (6) **not**
- (7) **and**
- (8) **or**
- (9) ==> ( *boolean implication*)

Operators appearing on a single line in this table are applied from left to right. For example,  $a-b-c$  is the same as  $(a-b)-c$ .

Here are some examples illustrating precedence, in which the left and right expressions are equivalent:

|                                  |                                    |
|----------------------------------|------------------------------------|
| $-1**2$                          | $-(1**2)$                          |
| $a+b*c$                          | $a+(b*c)$                          |
| $a*b/c$                          | $(a*b)/c$                          |
| $b \text{ or } c \text{ and } d$ | $b \text{ or } (c \text{ and } d)$ |
| $x < y \text{ and } y < z$       | $(x < y) \text{ and } (y < z)$     |

The final example illustrates the fact that in Turing, parentheses are not required when combining comparisons using **and** and **or**. These would be required in the Pascal language.

The type cheat operator # is applied after subscripting, subprogram calling, dotting, and ->. For example, in each of the following, # applies to the entire reference to the right.

#a(i)  
#r.y  
#p->x

The pointer following operator ^ is applied before subscripting, subprogram calling, dotting, and ->. For example, in the following, ^ applies to a, r and p.

^a(i)  
^r.y  
^p->x

Use parentheses to force ^ to apply to more of the reference. For example, in ^a(i), the ^ applies to a(i).

See also *infix* and *prefix* operators. See the **int**, **real**, **string**, **boolean**, **set**, **enum**, **char** and **char**(*n*) types.

## pred predecessor function

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>pred</b> ( <i>expn</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Description | The <b>pred</b> function accepts an integer, character, or an enumerated value and returns the integer minus one, the previous character, or the previous value in the enumeration. For example, <b>pred</b> ( 7 ) is 6.                                                                                                                                                                                                                                                                                                                                                                   |
| Example     | <p>This part of a Turing program fills up array <i>a</i> with the enumerated values <i>red</i>, <i>yellow</i>, <i>green</i>, <i>red</i>, <i>yellow</i>, <i>green</i>, etc.</p> <pre><b>type</b> colors : <b>enum</b> ( <i>green</i>, <i>yellow</i>, <i>red</i> )<br/><b>var</b> a : <b>array</b> 1 .. 100 <b>of</b> colors<br/><b>var</b> c : colors := colors . red<br/><b>for</b> i : 1 .. 100<br/>  a ( i ) := c<br/>  <b>if</b> c = colors . green <b>then</b><br/>    c := colors . red<br/>  <b>else</b><br/>    c := <b>pred</b> ( c )<br/>  <b>end if</b><br/><b>end for</b></pre> |
| Details     | It is illegal to apply <b>pred</b> to the first value of an enumeration.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| See also    | <b>succ</b> , <b>lower</b> and <b>upper</b> functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## prefix operator

|        |                                    |                                                                   |
|--------|------------------------------------|-------------------------------------------------------------------|
| Syntax | A <i>prefixOperator</i> is one of: |                                                                   |
|        | (a)                                | <b>+</b> % Integer and real identity<br>% (does not change value) |
|        | (b)                                | <b>-</b> % Integer and real negation                              |
|        | (c)                                | <b>not</b> % Not (Boolean negation)                               |
|        | (d)                                | <b>#</b> % Type cheat                                             |
|        | (e)                                | <b>^</b> % Pointer following                                      |

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | <p>A <i>prefix operator</i> is placed before a value or <i>operand</i> to produce another value. For example, if the value of <math>x</math> is seven, then <math>-x</math> is negative seven. There are also infix operators such as multiplication (<math>*</math>) and addition (<math>+</math>), which are placed between two values to produce a third value. See <i>infix operator</i>.</p> <p>The <math>+</math> and <math>-</math> prefix operators can be applied only to numeric values (integer, real and natural numbers). The <b>not</b> prefix can be applied only to true/false (boolean) values. For example <b>not</b> (<math>x &gt; y</math>) is equivalent to <math>x \leq y</math>. The <b>not</b> operator produces <b>true</b> from <b>false</b> and <b>false</b> from <b>true</b>.</p> <p>The <math>\#</math> operators is a type cheat (see <b>cheat</b>), and the <math>\wedge</math> operator is pointer following (see <b>pointer</b>).</p> |
| See also    | <b>int</b> , <b>real</b> and <b>boolean</b> types, as well as <i>precedence</i> (for the order of applying operators) and <i>infix operators</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## procedure declaration

|             |                                                                                                                                                                                                   |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>procedureDeclaration</i> is:</p> <pre><b>procedure</b> <i>id</i> [(<i>paramDeclaration</i> {, <i>paramDeclaration</i> })]     <i>statementsAndDeclarations</i> <b>end</b> <i>id</i></pre> |
| Description | A procedure declaration creates (but does not run) a new procedure. The name of the procedure ( <i>id</i> ) is given in two places, just after <b>procedure</b> and just after <b>end</b> .       |

### Example

```
procedure greetings
 put "Hello world"
end greetings

greetings % This outputs Hello world

procedure sayItAgain (msg : string, n : int)
 for i : 1 .. n
 put msg
 end for
end sayItAgain

sayItAgain ("Toot", 2) % Toot is output twice

procedure double (var x : real)
 x := 2 * x
```

```
end double
```

```
var y : real := 3.14
 double (y) % This doubles the value of y
```

Details            The set of parameters declared with the procedure are called *formal* parameters. In the *double* procedure, for example, *x* is a formal parameter. A procedure is called (invoked) by a *procedure call statement* which consists of the procedure's name followed by the parenthesized list of *actual* parameters (if any). For example, *double(y)* is a call having *y* as an actual parameter. If there are no parameters (see the *greet* procedure above), the call does not have parentheses. The keyword **procedure** can be abbreviated to **proc**.

Ordinarily, a procedure *returns* (finishes and goes back to the place where it was called) by reaching its **end**. However, the **return** statement in a procedure causes it to return immediately. Note that **return** can also be used in the main program to cause it to halt immediately.

Only parameters declared using **var** may be changed in the procedure, for example, *x* is changed in the *double* procedure. The upper bounds of arrays and strings that are parameters may be declared to be an asterisk (\*). This means that the bound is that of the actual parameter. See *paramDeclaration* for details about parameters.

Procedures and functions cannot be declared inside other procedures and functions.

The syntax of a *procedureDeclaration* presented above has been simplified by leaving out the optional **import** list, **pre** condition, **init** clause, **post** condition and exception handler. The full syntax is

```
procedure [pervasive] id
 [([paramDeclaration {,paramDeclaration}])]
 [: deviceSpecification]
 [pre trueFalseExpn]
 [init id := expn {, id := expn}]
 [post trueFalseExpn]
 [exceptionHandler]
 statementsAndDeclarations
end id
```

A procedure must be declared before being called. To allow for mutually recursive procedures, there are **forward** declarations of procedures with later declaration of each procedure **body**. See **forward** and **body** declarations for explanations.

See also            **import** list, **pre** condition, **init** clause, **post** condition and *exceptionHandler* for explanations of these features. See **pervasive** for information on **pervasive** procedures. See **exceptionHandler**. The optional *deviceSpecification* is used only in procedures declared in monitors and is used to create an *interrupt handling procedure*. See **monitor** for details.

# procedureCall statement

Syntax A *procedureCall* is:

*procedureId* [ ( [ *expn* { , *expn* } ] ) ]

Description A procedure call is a statement that calls (invokes or activates) a **procedure**. If the procedure has parameters, a parenthesized list of expressions (*expns*) must follow the procedure's name (*procedureId*).

Example

```
procedure greet
 put "Hello"
end greet

greet % This is a call to the greet procedure

procedure times (var i : int, factor : int)
 i := factor * i
end times

var j : int
 times (j, 4) % Multiply j by 4
```

Details A parameter declared in the header of a procedure is a *formal* parameter. For example, *i* and *factor* above are formal parameters. Each expression in the call is an *actual* parameter. For example, *j* and 4 above are actual parameters.

If a formal parameter is declared using **var**, then the expression passed to that parameter must be a variable reference (so its value can potentially be changed by the procedure). This means, for example, that it would be illegal to pass *j*+3 as the first parameter to *times*. The variable reference and the formal parameter must have equivalent types (see *equivalence* for details).

Each actual parameter passed to a non-**var** formal parameter must be assignable to that parameter (see *assignability* for details). See also *procedureDeclaration*.

In this explanation of *procedureCall*, we have up to this point ignored the possibility of procedures exported from modules, monitors and classes. If the procedure is being called from outside of a module or monitor *M* from which it has been exported, the syntax of the *procedureCall* is:

*M* . *procedureId* [ ( [ *expn* { , *expn* } ] ) ]



In other words, the module or monitor name and a dot must precede the procedure's name. If the procedure is being called from outside of a class from which it has been exported, the syntax of the *procedureCall* is one of:

- (a) *classId* (*p*) . *procedureId* [ ( [ *expn* { , *expn* } ] ) ]
- (b) *p* -> *procedureId* [ ( [ *expn* { , *expn* } ] ) ]

In these, *p* must be a pointer value that locates an object in the class. Form (b) is a short form for form (a).

See also **class**.

## process declaration

Syntax A *processDeclaration* is:

```
process id [([paramDeclaration { , paramDeclaration }])]
 statementsAndDeclarations
end id
```

Description A process declaration is much like a procedure declaration, but is activated by a **fork** statement rather than by a call. The **fork** statement starts concurrent (parallel) execution of the process while the statements following the **fork** continue to execute.

Example This program initiates (forks) two concurrent processes, one of which repeatedly outputs *Hi* and the other *Ho*. The resulting output is an unpredictable sequence of *Hi*'s and *Ho*'s as *greetings* executes twice concurrently, one instance with *word* set to *Hi* and the other with *word* set to *Ho*.

```
process greetings (word : string)
 loop
 put word
 end loop
end greetings

fork greetings ("Hi")
 fork greetings ("Ho")
```

Details The **process** declaration creates a template for a process (a concurrent activity), which is activated by a **fork** statement. A process declaration can appear wherever a module declaration is allowed except that a process declaration is not allowed in a class. The declarations and statements in a process declaration are the same as those in a procedure.

See *paramDeclaration* for details about parameters. There is an anomaly in parameters to processes, that can lead to errors. In particular, non-**var** parameters that are non-scalars (such as strings and arrays) are passed by reference. The result is that the target of the reference may change value while the process is executing, which in turn means that the seemingly constant parameter is not really constant. For example, if the string variable *s* were passed to the *greetings* process and subsequently changed, the value of *greetings*' formal parameter would change.

The syntax of a *processDeclaration* presented above has been simplified by leaving out the optional stack size (*compileTimeExpn*), **import** list, **pre** condition, **init** clause, **post** condition and exception handler.

The full syntax is:

```
process [pervasive] id
 [([paramDeclaration {,paramDeclaration }])]
 [: compileTimeExpn]
 [pre trueFalseExpn]
 [init id := expn {, id := expn }]
 [post trueFalseExpn]
 [exceptionHandler]
 statementsAndDeclarations
end id
```

See **pervasive** for information on **pervasive** processes. The optional *compileTimeExpn* following the parameter list (if any) is used to specify the number of bytes for the process' stack.

See also **import** list, **pre** condition, **init** clause, **post** condition and *exceptionHandler* for explanations of these additional features.

## program     a (main) program

Syntax             A *program* is:

*statementsAndDeclarations*

Description        A Turing program consists of a list of statements and declarations.

Example            This is a complete Turing program. It outputs *Alan M. Turing*.

**put** "Alan M. Turing"

Example            This is a complete Turing program. It outputs a triangle of stars.

```
var stars : string := ""
loop
 put stars
```

```
stars := stars + "*"
end loop
```

Example            This is a complete Turing program. It outputs *Hello* once and *Goodbye* twice.

```
procedure sayItAgain (what : string, n : int)
 for i : 1 .. n
 put what
 end for
end sayItAgain

sayItAgain ("Hello", 1)
sayItAgain ("Goodbye", 2)
```

Details            In a program there can be many units (see **unit**), one of which is the program (called the main program), the others of which are modules, monitors and classes. The main program is optionally preceded by an **import** list, which lists the units that it uses.

See also            **import** list.

## put statement

Syntax            A *putStatement* is:

```
put [: fileNumber ,] putItem { , putItem } [..]
```

Description        The **put** statement outputs each of the *putItems*. Usually, a new line is started in the output after the final *putItem*. If the optional dot-dot (..) is present, though, subsequent output will be continued on the current output line. With character graphics, the omission of dot-dot causes the remainder of the output line to be cleared to blanks.

Ordinarily, the output goes to the screen. However, if the *fileNumber* is present, the output goes to the file specified by the file number (see the **open** statement for details). Also, output can be redirected from the screen to a file, in which case all **put** statements without a file number are sent to the file instead of the screen.

Example

```
var n : int := 5
put "Alice owes me $", n
 % Output is: Alice owes me $5
 % Note that no extra space is
 % output before an integer such as n.
```

## Example

| <u>Statement</u>                 | <u>Output</u> | <u>Notes</u>              |
|----------------------------------|---------------|---------------------------|
| <b>put</b> 24                    | 24            |                           |
| <b>put</b> 1/10                  | 0.1           | % Trailing zeros omitted  |
| <b>put</b> 100/10                | 10            | % Decimal point omitted   |
| <b>put</b> 5/3                   | 1.666667      | % 6 fraction digits       |
| <b>put</b> <i>sqrt</i> (2)       | 1.414214      | % 6 fraction digits       |
| <b>put</b> 4.86 * 10**9          | 4.86e9        | % Exponent for $\geq 1e6$ |
| <b>put</b> 121 : 5               | bb121         | % Width 5; b is blank     |
| <b>put</b> 1.37 : 6 : 3          | b1.370        | % Fraction width of 3     |
| <b>put</b> 1.37 : 11 : 3 : 2     | bb1.370e+00   | % Exponent width of 2     |
| <b>put</b> "Say \"Hello\""       | Say "Hello"   |                           |
| <b>put</b> "XX" : 4, "Y"         | XXbbY         | % Blank shown as b        |
| <b>put</b> true <b>and</b> false | false         | % Put out a boolean value |
| <b>put</b> 1 < 2                 | true          | % Put out a boolean value |

Example A single blank line is output this way:

```
put "" % Output null string then new line
```

This **put** statement is sometimes used to close off a line that has been output piece by piece using **put** with dot-dot.

Details The general form of a *putItem* is one of:

- (a) *expn* [:*widthExpn* [:*fractionWidth* [:*exponentWidth* ] ] ]
- (b) **skip**

See the above examples for uses of *widthExpn*, *fractionWidth* and *exponentWidth*. For the exact meaning of these three widths, see the definitions of the functions *realstr*, *frealstr* and *erealstr*. The **skip** item is used to end the current output line and start a new line.

Details The **put** semantics allow put's of enum values. The values printed are the element names themselves, case sensitive. For example:

```
type colors : enum (red, green, blue)
var c : colors := colors . red
put c % outputs "red" (without the quotes)
```

Details The **put** semantics allow put's of **boolean** values. The values printed are either "true" or "false" (without the quotes). For example:

```
var c : boolean := true or false
put c % outputs "true" (without the quotes)
```

# quit fail statement

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>quitStatement</i> is:</p> <p style="text-align: center;"><b>quit</b> [ <i>guiltyParty</i> ] [ : <i>quitReason</i> ]</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Description | <p>The <b>quit</b> statement causes a program (or concurrent process) to fail. The failure (called an <i>exception</i>) either aborts the program (or process) or causes control to be passed to an exception handler.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Example     | <p>In the <i>inputLines</i> procedure, halt the program if end of file is encountered before the string "stop" is read. Note that a <b>return</b> statement in the procedure would terminate the procedure but not the entire program.</p> <pre>var line : array 1 .. 50 of string  procedure inputLines   var i : int := 0   loop     if eof then       put "Missing 'stop' in input"       quit      % Halt entire program     end if     i := i + 1     get line ( i )     exit when line ( i ) = "stop"   end loop end inputLines  inputLines</pre>                                                                                                                                                                                                              |
| Details     | <p>In the simple case, the optional <i>guiltyParty</i> and <i>quitReason</i> are omitted. The <i>guiltyParty</i> option is used to specify the position of failure. See <i>exceptionHandler</i> for an example of a <b>quit</b> statement used in conjunction with a handler. A handler, which is located at the beginning of a subprogram body, is given control when a <b>quit</b> is executed or a failure, such as division by zero, occurs in the subprogram.</p> <p>The <i>guiltyParty</i> option is used to designate the location of the failure, for example, to tell the debugger what line is considered to be the location of the failure. A <i>guiltyParty</i> is one of:</p> <ul style="list-style-type: none"><li>(a) &lt;</li><li>(b) &gt;</li></ul> |

If *guiltyParty* is omitted, the failure is considered to occur at the **quit** statement. If it is *<*, the failure is considered to occur at the call to the present subprogram. For example, if the present subprogram implements square root *sqr*t and is passed a negative argument, it can use *<* to specify that the caller provided a faulty argument. If *guiltyParty* is *>*, this means the failure has already occurred and is being passed on to the next handler or to the system. To summarize, the three possibilities for designating the location of the failure are:

- (a)     *<*           Caller is cause of failure
- (b)     *>*           The exception being handled is the cause.
- (c)     (omitted *guiltyParty*) The present **quit** is the cause.

The *quitReason* is an integer expression which is used to identify the kind of failure. If it is omitted, a default value is chosen in the following manner. If *guiltyParty* is omitted or is *<*, the default is 1. If *guiltyParty* is *>* and an exception handler is active, the default is the *quitReason* of the exception being handled. If no exception is being handled, the default is 1. In the case of program abortion, the implementation may pass the *quitReason* to the operating system or programming environment.

See also     *exceptionHandler*, **return** and **result**.

## Rand All

|              |                                                                                                                                                                                                         |                                                             |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that deal with random numbers.</p> <p>All routines in the <b>Rand</b> unit are exported qualified (and thus must be prefaced with "<b>Rand.</b>").</p> |                                                             |
| Entry Points | <b>Real</b>                                                                                                                                                                                             | Returns a random real number.                               |
|              | <b>Int</b>                                                                                                                                                                                              | Returns a random integer.                                   |
|              | <b>Reset</b>                                                                                                                                                                                            | Sets the seed in the default sequence to a default value.   |
|              | <b>Set</b>                                                                                                                                                                                              | Sets the seed in the default sequence to a specified value. |
|              | <b>Next</b>                                                                                                                                                                                             | Returns a random real number from a sequence.               |
|              | <b>Seed</b>                                                                                                                                                                                             | Sets a seed in a sequence.                                  |

## rand random real number procedure

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>rand ( var <i>r</i> : real )</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description | The <b>rand</b> statement is used to create a pseudo-random number in the range zero to one. For example, if <i>x</i> is a real number, after <b>rand</b> ( <i>x</i> ), <i>x</i> would have a value such as 0.729548 or 0.352879.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Example     | <p>This program repeatedly and randomly prints out <i>Hi ho, hi ho</i> or <i>It's off to work we go</i>.</p> <pre>var r : real loop   rand ( r )   if r &gt; 0.5 then     put "Hi ho, hi ho"   else     put "It's off to work we go"   end if end loop</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Details     | <p>The <b>rand</b> statement sets its parameter to the next value of a sequence of pseudo-random real numbers that approximates a uniform distribution over the range <math>0 &lt; r &lt; 1</math>.</p> <p>Each time a program runs, <b>rand</b> uses the same pseudo-random number sequence. To get a different sequence (actually, to start the sequence at a different point), use the <b>randomize</b> procedure.</p> <p>To use several sequences of repeatable pseudo-random number sequences, use the <b>randseed</b> and <b>randnext</b> procedures.</p> <p>In many languages, <b>rand</b> would be a function rather than a procedure. It has been designed as a procedure in Turing to respect the mathematical idea that every call to a function using the same arguments (or no arguments at all) should return the same value. If <b>rand</b> were a function, this would not be true.</p> |
| See also    | <b>randint</b> , <b>randomize</b> , <b>randseed</b> and <b>randnext</b> .<br>See also predefined unit <b>Rand</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

# Rand.Int All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Rand.Int</b> ( <i>low</i> , <i>high</i> : <b>int</b> ) : <b>int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Description | The <b>Rand.Int</b> statement is used to create a pseudo-random integer in the range <i>low</i> to <i>high</i> , inclusive. For example, if <i>i</i> is an integer, after <i>i</i> := <b>Rand.Int</b> ( <i>i</i> , 1, 10), <i>i</i> would have a value such as 7 or 2 or 10.                                                                                                                                                                                                                                                                                                                                                  |
| Example     | This program simulates the repeated rolling of a six sided die.<br><pre>loop   put "Rolled ", Rand.Int (1, 6) end loop</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Details     | <p>The <b>Rand.Int</b> statement sets its parameter to the next value of a sequence of pseudo-random integers that approximates a uniform distribution over the range <math>low \leq i \leq high</math>. It is required that <math>low \leq high</math>.</p> <p>Each time a program runs, <b>Rand.Int</b> uses a different pseudo-random number sequence. To always get the same sequence (actually, to start the sequence at the same point), use the <b>Rand.Set</b> procedure.</p> <p>To use several sequences of repeatable pseudo-random number sequences, use the <b>Rand.Seed</b> and <b>Rand.Next</b> procedures.</p> |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Rand.Int</b> , not by calling <b>Int</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| See also    | <b>Rand.Real</b> , <b>Rand.Set</b> , <b>Rand.Seed</b> and <b>Rand.Next</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

# Rand.Next All

|        |                                                         |
|--------|---------------------------------------------------------|
| Syntax | <b>Rand.Next</b> ( <i>seq</i> : 1 .. 10 ) : <b>real</b> |
|--------|---------------------------------------------------------|



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | <p>The <b>Rand.Next</b> procedure is used when you need several sequences of pseudo-random numbers, and you need to be able to exactly repeat these sequences for a number of simulations. The <b>Rand.Next</b> procedure is the same as <b>rand</b>, except <i>seq</i> specifies one of ten independent and repeatable sequences of pseudo-random real numbers.</p> <p>The <b>Rand.Seed</b> procedure is used to start one of these sequences at a particular point.</p> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Rand.Next</b>, not by calling <b>Next</b>.</p>                                                                                                                                                                                                                                                                                                                                 |
| See also    | <b>Rand.Seed</b> , <b>Rand.Int</b> , <b>Rand.Real</b> and <b>Rand.Next</b> .                                                                                                                                                                                                                                                                                                                                                                                              |

## Rand.Real All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Rand.Real : real</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Description | <p>The <b>Rand.Real</b> function returns a pseudo-random number in the range zero to one. For example, if <i>x</i> is a real number, after <i>x</i> := <b>Rand.Real</b>, <i>x</i> would have a value such as 0.729548 or 0.352879.</p>                                                                                                                                                                                                                                                                                                                                        |
| Example     | <p>This program repeatedly and randomly prints out <i>Hi ho</i>, <i>hi ho</i> or <i>It's off to work we go</i>.</p> <pre> loop   if Rand.Real &gt; 0.5 then     put "Hi ho, hi ho"   else     put "It's off to work we go"   end if end loop </pre>                                                                                                                                                                                                                                                                                                                           |
| Details     | <p>The <b>Rand.Real</b> function sets its parameter to the next value of a sequence of pseudo-random real numbers that approximates a uniform distribution over the range <math>0 &lt; r &lt; 1</math>.</p> <p>Each time a program runs, <b>Rand.Real</b> uses a different pseudo-random number sequence. To always get the same sequence (actually, to start the sequence at the same point), use the <b>Rand.Set</b> procedure.</p> <p>To use several sequences of repeatable pseudo-random number sequences, use the <b>Rand.Seed</b> and <b>Rand.Next</b> procedures.</p> |

|          |                                                                                                                                  |
|----------|----------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Rand.Real</b> , not by calling <b>Real</b> . |
| See also | <b>Rand.Int</b> , <b>Rand.Set</b> , <b>Rand.Seed</b> and <b>Rand.Next</b> .                                                      |

## Rand.Reset All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Rand.Reset</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Description | This is a procedure with no parameters that resets the sequences of pseudo-random numbers produced by <b>Rand.Real</b> and <b>Rand.Int</b> . This allows identical executions of the same program to produce identical results.                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Example     | <p>This program simulates the repeated rolling of a six sided die. Each time the program runs, the same sequence of rolls occurs.</p> <pre> <b>Rand.Reset</b> loop   put "Rolled ", <b>Rand.Int</b> (1, 6) end loop </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Details     | <p>If <b>Rand.Reset</b> and <b>Rand.Set</b> are not used, each time a program runs <b>Rand.Real</b> and <b>Rand.Int</b> use a different pseudo-random number sequence. To get the same sequence each time (actually, to start the sequence at a different point), use <b>Rand.Reset</b> or <b>Rand.Set</b>.</p> <p>The <b>Rand.Reset</b> procedure can be called any time. However, to make it work, it should only be called once per program. Any call to <b>Rand.Reset</b> after the first one is ignored.</p> <p>To use several sequences of repeatable pseudo-random number sequences, use the <b>Rand.Seed</b> and <b>Rand.Next</b> procedures.</p> |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Rand.Reset</b> , not by calling <b>Reset</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| See also    | <b>Rand.Set</b> , <b>Rand.Int</b> , <b>Rand.Real</b> , <b>Rand.Seed</b> and <b>Rand.Next</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Rand.Seed All

|             |                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Rand.Seed</b> ( <i>seed</i> : <b>nat4</b> , <i>seq</i> : 1 .. 10 )                                                                                                                                     |
| Description | The <b>Rand.Seed</b> procedure restarts one of the sequences generated by <b>Rand.Next</b> . Each restart with the same seed causes <b>Rand.Next</b> to produce the same sequence for the given sequence. |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Rand.Seed</b> , not by calling <b>Seed</b> .                                                                          |
| See also    | <b>Rand.Next</b> , <b>Rand.Int</b> , <b>Rand.Real</b> , and <b>Rand.Set</b> .                                                                                                                             |

## Rand.Set All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Rand.Set</b> ( <i>seed</i> : <b>nat4</b> )                                                                                                                                                                                                                                                                                                                                                                                                           |
| Description | This procedure sets the seed for sequences of pseudo-random numbers produced by <b>Rand.Real</b> and <b>Rand.Int</b> . This allows identical executions of the same program to produce identical results.                                                                                                                                                                                                                                               |
| Example     | <p>This program simulates the repeated rolling of a six sided die. Each time the program runs, the same sequence of rolls occurs.</p> <pre><b>Rand.Set</b> (16#1234ABCD) <b>loop</b>   <b>put</b> "Rolled ", <b>Rand.Int</b> (1, 6) <b>end loop</b></pre>                                                                                                                                                                                               |
| Details     | <p>If <b>Rand.Reset</b> and <b>Rand.Set</b> are not used, each time a program runs <b>Rand.Real</b> and <b>Rand.Int</b> use a different pseudo-random number sequence. To get the same sequence each time (actually, to start the sequence at a different point), use <b>Rand.Reset</b> or <b>Rand.Set</b>.</p> <p>To use several sequences of repeatable pseudo-random number sequences, use the <b>Rand.Seed</b> and <b>Rand.Next</b> procedures.</p> |

|          |                                                                                                                                |
|----------|--------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Rand.Set</b> , not by calling <b>Set</b> . |
| See also | <b>Rand.Reset</b> , <b>Rand.Int</b> , <b>Rand.Real</b> , <b>Rand.Seed</b> and <b>Rand.Next</b> .                               |

## randomize procedure

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>randomize</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Description | This is a procedure with no parameters that resets the sequences of pseudo-random numbers produced by <b>rand</b> and <b>randint</b> . This allows different executions of the same program to produce different results.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Example     | <p>This program simulates the repeated rolling of a six sided die. Each time the program runs, a different sequence of rolls occurs (unless there is quite a coincidence or you run the program a lot of times!).</p> <pre> <b>randomize</b> <b>var</b> roll : <b>int</b> <b>loop</b>   <b>rand</b> ( i )   <b>put</b> "Rolled ", i <b>end loop</b> </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Details     | <p>If <b>randomize</b> is not used, each time a program runs, <b>rand</b> and <b>randint</b> use the same pseudo-random number sequences. To get a different sequence (actually, to start the sequence at a different point), use <b>randomize</b>.</p> <p><b>randomize</b> on the PC and Mac initialize the random sequence based on the time of day (among other things). If <b>randomize</b> is called more than once in the same second, then it will restart the sequence with the same random number. Consequently, one should <b>not</b> place <b>randomize</b> in a loop. It should be called once per program at or near the beginning of the program.</p> <p>To use several sequences of repeatable pseudo-random number sequences, use the <b>randseed</b> and <b>randnext</b> procedures.</p> |
| See also    | <p><b>randint</b>, <b>rand</b>, <b>randseed</b> and <b>randnext</b>.</p> <p>See also predefined unit <b>Rand</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

# read file statement

Dangerous parts

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>readStatement</i> is:</p> <pre><b>read</b> : <i>fileNumber</i> [ : <i>status</i> ] , <i>readItem</i> { ,<i>readItem</i> }</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Description | <p>The <b>read</b> statement inputs each of the <i>readItems</i> from the specified file. These items are input directly using the <i>binary</i> format that they have on the file. In other words, the items are not in source (ASCII or EBCDIC) format. In the common case, these items have been output to the file using the <b>write</b> statement.</p> <p>By contrast, the <b>get</b> and <b>put</b> statements use source format, which a person can read using an ordinary text editor.</p>                                                                                                                                                                                                                                                                                                                                                                                                  |
| Example     | <p>This example shows how to input a complete employee record using a <b>read</b> statement.</p> <pre><b>var</b> <i>employeeRecord</i> :<br/>    <b>record</b><br/>        <i>name</i> : <b>string</b> ( 30 )<br/>        <i>pay</i> : <b>int</b><br/>        <i>dept</i> : 0 .. 9<br/>    <b>end record</b><br/><b>var</b> <i>fileNo</i> : <b>int</b><br/><b>open</b> : <i>fileNo</i>, "payroll", <b>read</b><br/>...<br/>        <b>read</b> : <i>fileNo</i>, <i>employeeRecord</i></pre>                                                                                                                                                                                                                                                                                                                                                                                                          |
| Details     | <p>The <i>fileNumber</i> must specify a file that is open with <b>read</b> capability (or a program argument file that is implicitly opened).</p> <p>The optional <i>status</i> is an <b>int</b> variable that is set to implementation-dependent information about the <b>read</b>. If <i>status</i> is returned as zero, the read was successful. Otherwise <i>status</i> gives information about the incomplete or failed <b>read</b> (which is not documented here). You commonly use <i>status</i> when you are reading a record or array from a file and you are not sure if the entire item exists on the file. If it does not exist, the <b>read</b> will fail part way through, but your program can continue and diagnose the problem by inspecting <i>status</i>.</p> <p>A <i>readItem</i> is:</p> <pre><i>variableReference</i> [ : <i>requestedSize</i> [ : <i>actualSize</i> ] ]</pre> |

Each *readItem* specifies a variable to be read in internal form. The optional *requestedSize* is an integer value giving the number of bytes of data to be read. The *requestedSize* should be less than or equal to the size of the item's internal form in memory (else a warning message is issued). If no *requestedSize* is given, the size of the item in memory is used. The optional *actualSize* is an **int** variable that is set to the number of bytes actually read.

An array, record or union may be read and written as a whole.

It is dangerous to read into pointer variables, as this allows the possibility of creating incorrect addresses in the pointers. It is also dangerous to read more bytes than are in the *readItem*.

See also the **write**, **open**, **close**, **seek**, **tell**, **get** and **put** statements.

## real the real number type

Syntax **real**

Description The **real** number type is used for numbers that have fractional parts, for example, 3.14159. Real numbers can be combined by various operators such as addition (+) and multiplication (\*). Real numbers can also be combined with integers (whole numbers, such as 23, 0 and -9), in which case the result is generally a real number. An integer can always be assigned to a real variable, with implicit conversion to **real**.

Example

```
var weight, x : real
var x : real := 9.83
var tax := 0.7 % The type is implicitly real because
 % 0.7 is a real number
```

Details See also *explicitRealConstant*. The **int** type is used instead of **real**, when values are whole numbers. See **int** for details.

Real numbers can be converted to integers using **ceil** (ceiling), **floor**, or **round**. Real numbers can be converted to strings using **erealstr**, **frealstr**, and **realstr**. These conversion functions correspond exactly to the formatting used for the **put** statement with real numbers. Strings can be converted to real numbers using **streal**. See descriptions of these conversion functions.

The predefined functions for real numbers include **min**, **max**, **sqrt**, **sin**, **cos**, **arctan**, **sind**, **cosd**, **arcand**, **ln** and **exp**. See the descriptions of these functions.

Pseudo-random sequences of real numbers can be generated using **rand**. See the description of this procedure.

The Turing Report gives a formal definition (not repeated here) of implemented real numbers in terms of their required accuracy relative to infinitely accurate (mathematical) real numbers.

Turing implements real numbers using 8 byte floating point representation. This provides 14 to 16 decimal digits of precision and an exponent range of at least -38 .. 38. The PC and Macintosh versions of Turing have 16 decimal digits of accuracy because they use IEEE standard floating point representation.

See also **real***n*.

## **real***n*    n-byte real number type

### Syntax

- (a)    **real4**            % 4-byte real number
- (b)    **real8**            % 8-byte real number

**Description**    The **real***n* (*n*-byte real number) types are machine-dependent types that occupy a specified number of bytes. By contrast, the **real** type is, in principle, a machine-independent and mathematical type (however, it overflows when the exponent of the value is too large or small and it has only a limited amount of precision).

### Example

```
var width : real4
 height : real8
```

**Details**    Turing implements the type **real** using 8 byte floating point representation. This provides 14 to 16 decimal digits of precision and an exponent range of at least -38 .. 38. The PC and Macintosh versions of Turing have 16 decimal digits of accuracy because they use IEEE standard floating point representation.

This implies that **real8** and **real** are essentially the same type, so in practice there is no advantage to using **real8** rather than **real**. However, **real4** has the advantage of occupying half as much space (with correspondingly reduced precision).

Arithmetic for all real types (**real**, **real4** and **real8**) is carried out with the accuracy and exponent range of 8-byte reals.

The type **real4** is sometimes called *single precision* (because it occupies a single 4-byte word) and **real8** is sometimes called *double precision*.

## realstr    real-to-string function

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>realstr</b> ( <i>r</i> : <b>real</b> , <i>width</i> : <b>int</b> ) : <b>string</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Description | <p>The <b>realstr</b> function is used to convert a <b>real</b> number to a string. For example, <b>realstr</b> (2.5e1, 4)="bb25" where <i>b</i> represents a blank. The string is an approximation to <i>r</i>, padded on the left with blanks as necessary to a length of <i>width</i>.</p> <p>The <i>width</i> parameter must be non-negative. If the <i>width</i> parameter is not large enough to represent the value of <i>r</i> it is implicitly increased as needed. The displayed value is rounded to the nearest decimal equivalent with this accuracy. In the case of a tie, the display value is rounded to the next larger value.</p> <p>The string <b>realstr</b> (<i>r</i>, <i>width</i>) is the same as the string <b>frealstr</b> (<i>r</i>, <i>width</i>, <i>defaultfw</i>) when <i>r</i> = 0 or when <math>1e-3 &lt; \text{abs}(r) &lt; 1e6</math>, otherwise the same as <b>erealstr</b> (<i>r</i>, <i>width</i>, <i>defaultfw</i>, <i>defaultew</i>), with the following exceptions. With <i>realstr</i>, trailing fraction zeroes are omitted, and the decimal point is omitted if the entire fraction is zero. (These omissions take place even if the exponent part is printed.) If an exponent is printed, any plus sign and leading zeroes are omitted. Thus, whole number values are in general displayed as integers.</p> <p><i>Defaultfw</i> is an implementation-defined number of fractional digits to be displayed. For most implementations, <i>defaultfw</i> will be 6.</p> <p><i>Defaultew</i> is an implementation-defined number of exponent digits to be displayed. For most implementations, <i>defaultew</i> will be 2.</p> <p>The <b>realstr</b> function approximates the inverse of <b>strreal</b>, although round-off errors keep these from being exact inverses.</p> |
| See also    | the <b>erealstr</b> , <b>frealstr</b> , <b>strreal</b> , <b>intstr</b> and <b>strint</b> functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |



# record type

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>recordType</i> is:</p> <pre><b>record</b>     <i>id</i> {, <i>id</i> } : <i>typeSpec</i>     { <i>id</i> {, <i>id</i> } : <i>typeSpec</i> } <b>end record</b></pre>                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Description | <p>Each value of a record type consists of fields, one field for each name (<i>id</i>) declared inside the record. In the following example, the fields are <i>name</i>, <i>phoneNumber</i> and <i>address</i>.</p>                                                                                                                                                                                                                                                                                                                                                                              |
| Example     | <pre><b>type</b> <i>phoneRecord</i> :     <b>record</b>         <i>name</i> : <b>string</b> ( 20 )         <i>phoneNumber</i> : <b>int</b>         <i>address</i> : <b>string</b> ( 50 )     <b>end record</b> ... <b>var</b> <i>oneEntry</i> : <i>phoneRecord</i> <b>var</b> <i>phoneBook</i> : <b>array</b> 1 .. 100 <b>of</b> <i>phoneRecord</i> <b>var</b> <i>i</i> : <b>int</b> <i>oneEntry</i> .<i>name</i> := "Turing, Alan" ...         <i>phoneBook</i> ( <i>i</i> ) := <i>oneEntry</i>           % Assign whole record</pre>                                                           |
| Details     | <p>In a record, <i>id</i>'s of fields must be distinct. However, these need not be distinct from identifiers outside the record. Records can be assigned as a whole (to records of an equivalent type), but they cannot be compared. A semicolon can optionally follow each <i>typeSpec</i>.</p> <p>Any array contained in a record must have bounds that are known at compile time.</p> <p>The notation <i>-&gt;</i> can be used to access record fields. For example, if <i>p</i> is a pointer to <i>phoneRecord</i>, <i>p-&gt;name</i> locates the <i>name</i> field. See <b>pointer</b>.</p> |

## register      use machine register      Dirty

**Description**      When a variable, constant or parameter is declared, you can request that the item be placed in a machine register. This should be done only for programs requiring considerable efficiency.

**Example**

```
var register counter : int
const register maxCounter : int := 100
procedure p (register x : real)
...
end p
```

**Details**      Items can be requested to be in registers only if they are local to a subprogram (not global variables, declared in the main program, a module, monitor or class). Items requested to be in registers cannot be bound to, passed to reference parameters, have their address taken by **addr**, or have certain type cheats applied to them (since a machine register has no address).

The request to use a register may be ignored. For example, the current (1999) interpretive implementation uses pseudo-code, which has no machine registers, and so ignores the **register** keyword. For the syntax of using this keyword, see **var** declaration, **const** declaration and **paramDeclaration**.

## rem      remainder operator

**Syntax**      **rem**

**Description**      The **rem** (*remainder*) operator produces the remainder of one number divided by another. For example, 7 **rem** 2 produces 1 and -12 **rem** 5 produces -2.

**Example**      In this example, *eggCount* is the total number of eggs. The first **put** statement determines how many dozen eggs there are. The second **put** statement determines how many extra eggs there are beyond the last dozen.

```

var eggCount : int
get eggCount
put "You have ", eggCount div 12, " dozen eggs"
 put "You have ", eggCount rem 12, " left over"

```

See also *infix operators, precedence of operators* and the **mod** and **div** operators.

## repeat    make copies of string function

|             |                                                                                                                                                                                                                                                             |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>repeat</b> ( <i>s</i> : <b>string</b> , <i>i</i> : <b>int</b> ) : <b>string</b>                                                                                                                                                                          |
| Description | The <b>repeat</b> function returns <i>i</i> copies of string <i>s</i> catenated together. For example, <b>repeat</b> ("X", 4) is XXXX.                                                                                                                      |
| Example     | This program outputs <i>HoHoHo</i> .<br><pre> var word : <b>string</b> := "Ho"     put repeat ( word, 3 ) </pre>                                                                                                                                            |
| Details     | If <i>i</i> is less than or equal to zero, the null string "" is returned. The <b>repeat</b> function is often used for spacing of output. For example, this statement skips 20 blanks before outputting <i>x</i> .<br><pre> put repeat (" ", 20), x </pre> |

## result    statement

|             |                                                                                                                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | A <i>resultStatement</i> is:<br><b>result</b> <i>expn</i>                                                                                                                                                                  |
| Description | A <b>result</b> statement, which must appear only in a <b>function</b> , is used to provide the value of the function.                                                                                                     |
| Example     | This function doubles its parameter.<br><pre> <b>function</b> double ( <i>x</i> : <b>real</b> ) : <b>real</b>     <b>result</b> 2 * <i>x</i> <b>end</b> double     put double ( 5.3 )           % This outputs 10.6 </pre> |

Example        This function finds the position of a name in a list.

```
function find (a : array 1 .. 100 of string) : int
 for i : 1 .. 100
 if a (i) = name then
 result i
 end if
 end for
end find
```

Details        The execution of a **result** statement computes the value of the expression (*expn*) and terminates the function, returning the value as the value of the function.

The expression must be assignable to the result type of the function, for example, in *double*, *2\*x* is assignable to **real**. (See the *assignmentStatement* for the definition of assignable.)

A function must terminate by executing a **result** statement and not by reaching the end of the function.

## return      statement

Syntax        A *returnStatement* is:

**return**

Description    A **return** statement terminates the **procedure** (or main program) in which it appears. Ordinarily, a procedure (or main program) terminates by reaching its end; the **return** statement is used to cause early termination.

Example        This procedure takes no action if the *errorHasOccurred* flag has been set to true.

```
procedure double
 if errorHasOccurred then
 return % Terminate this procedure
 end if
 ... handle usual case in this procedure ...
end double
```

Details        A **return** must not appear as a statement in (the outermost level of) a module, nor can it appear in a **function**.

# RGB

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description  | <p>This unit contains the predefined constants for the basic colors and the subprograms to change the color palette.</p> <p>All subprograms in the <b>RGB</b> unit are exported qualified (and thus must be prefaced with "<b>RGB.</b>"). All the color constants are exported unqualified and thus do not need the <b>RGB</b> prefix.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Entry Points | <p><b>black, blue, green, cyan, red, magenta,</b> color names constants<br/> <b>purple, brown, white, gray, grey, brightblue,</b> (exported unqualified)<br/> <b>brightgreen, brightcyan, brightred,</b><br/> <b>brightmagenta, brightpurple, yellow,</b><br/> <b>brightwhite, darkgray, darkgrey, colorfg,</b><br/> <b>colourfg, colorbg, colourbg</b></p> <p><b>GetColor</b> Gets the current red, green and blue values of a specified color number.</p> <p><b>GetColour</b> Gets the current red, green and blue values of a specified color number.</p> <p><b>SetColor</b> Sets the red, green and blue values of a specified color number.</p> <p><b>SetColour</b> Sets the red, green and blue values of a specified color number.</p> <p><b>AddColor</b> Creates a new color number with a specified red, green and blue value.</p> <p><b>AddColour</b> Creates a new color number with a specified red, green and blue value.</p> |

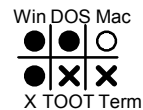
## RGB.AddColor



|        |                                                                                        |
|--------|----------------------------------------------------------------------------------------|
| Syntax | <b>RGB.AddColor</b> ( <i>redComp, greenComp, blueComp</i> : <b>real</b> ) : <b>int</b> |
|--------|----------------------------------------------------------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | <p>The <b>RGB.AddColor</b> function attempts to create a new color with the red, green and blue components specified. If successful, the function returns a new color number (usually one greater than <b>maxcolor</b>) and <b>maxcolor</b> is updated by adding 1 to it. If it is unsuccessful, the function returns -1 and <b>Error.Last</b> and <b>Error.LastMsg</b> can be used to determine the cause of the problem.</p> <p>The red, green and blue values must normalized to be between 0 and 1. Thus to add the pure red to the color palette, you would call:</p> <p style="text-align: center;"><i>newColor</i> := <b>RGB.AddColor</b> (1.0, 0.0, 0.0)</p> <p><i>newColor</i> would be set to the color added, or -1 if the attempt to add a color failed.</p> |
| Example     | <p>This program adds a palette of 16 blues to the end of the color palette.</p> <pre> var clr : int for blueShade : 0 .. 15   clr = RGB.AddColor (0, 0, blueShade / 15)   if clr = -1 then     put "Color add failed on shade number ", blueShade     exit   else     put "Added color number ", clr   end if end for </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Details     | <b>RGB.AddColour</b> is an alternate spelling for <b>RGB.AddColor</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>RGB.AddColor</b>, not by calling <b>AddColor</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| See also    | <b>RGB.GetColor</b> and <b>RGB.SetColor</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## RGB.GetColor



|             |                                                                                                                                                                                                                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>RGB.GetColor</b> ( <i>colorNumber</i> : <b>int</b> ,<br><b>var</b> <i>redComp</i> , <i>greenComp</i> , <i>blueComp</i> : <b>real</b> )                                                                                                                                                                                                                           |
| Description | <p>The <b>RGB.GetColor</b> procedure returns the red, green and blue components to the color associated with the <i>colorNumber</i> parameter. The red, green and blue values are normalized to be between 0 and 1. Thus color white returns 1.0 for the <i>redComp</i>, <i>greenComp</i> and <i>blueComp</i> values and color black returns 0.0 for all three.</p> |

Example      This program gets the components of all the available colors.

```
put "Color Red Green Blue"
for clr : 0 .. maxcolor
 var redComp, greenComp, blueComp : int
 RGB.GetColor (clr, redComp, greenComp, blueComp)
 put clr : 4, " ", redComp : 6 : 4, " ", greenComp : 6 : 4, " ",
 blueComp : 6 : 4
end for
```

Details      **RGB.GetColour** is an alternate spelling for **RGB.GetColor**.

Status      Exported qualified.  
This means that you can only call the function by calling **RGB.GetColor**,  
not by calling **GetColor**.

See also      **RGB.SetColor** and **RGB.AddColor**.

## RGB.maxcolor



Syntax      **maxcolor : int**

Description      The **maxcolor** function is used to determine the maximum color number for the current mode of the screen. The alternate spelling is **maxcolour**.

Example      This program outputs the maximum color number.

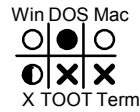
```
setscreen ("graphics")
...
put "The maximum color number is ", maxcolor
```

Details      The screen should be in a "screen" or "graphics" mode. If it is not, it will automatically be set to "screen" mode. See **View.Set** for details.  
For IBM PC compatibles in "screen" mode, **maxcolor** = 15. For the default IBM PC compatible "graphics" mode (CGA), **maxcolor** = 3.

Status      Exported unqualified.  
This means that you can call the function by calling **maxcolor** or by calling **RGB.maxcolor**.

See also      **Draw.Dot** for examples of the use of **maxcolor**. See the **Text.Color** procedure which is used for setting the currently-active color.

# RGB.SetColor



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>RGB.SetColor</b> ( <i>colorNumber</i> : <b>int</b> ,<br><i>redComp</i> , <i>greenComp</i> , <i>blueComp</i> : <b>real</b> )                                                                                                                                                                                                                                                                                                                                                                                                             |
| Description | <p>The <b>RGB.SetColor</b> function sets the red, green and blue components of the color associated with the <i>colorNumber</i> parameter. The red, green and blue values must normalized to be between 0 and 1. Thus to set the color associated with the <i>colorNumber</i> parameter to pure red, you would call:</p> <p style="text-align: center;"><b>RGB.SetColor</b> (<i>colorNumber</i>, 1.0, 0.0, 0.0)</p> <p>It is wise to use <b>Error.Last</b> and <b>Error.LastMsg</b> to check to see if the color change is successful.</p> |
| Example     | <p>This program sets all the available colors to shades of red</p> <pre>for clr : 0 .. maxcolor   if not RGB.SetColor (clr, clr / maxcolor, 0, 0) then     put "Color set failed on color number ", clr     exit   end if end for</pre>                                                                                                                                                                                                                                                                                                    |
| Details     | <b>RGB.SetColour</b> is an alternate spelling for <b>RGB.SetColor</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>RGB.SetColor</b>, not by calling <b>SetColor</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                           |
| See also    | <b>RGB.GetColor</b> and <b>RGB.AddColor</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## round    real-to-integer function

|        |                                                      |
|--------|------------------------------------------------------|
| Syntax | <b>round</b> ( <i>r</i> : <b>real</b> ) : <b>int</b> |
|--------|------------------------------------------------------|



|             |                                                                                                                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | The <b>round</b> function is used to convert a <b>real</b> number to an integer. The result is the nearest integer to $r$ . In the case of a tie, the numerically larger value is returned. For example, <b>round</b> (3) is 3, <b>round</b> (2.85) is 3 and <b>round</b> (-8.43) is -8. |
| See also    | the <b>floor</b> and <b>ceil</b> functions.                                                                                                                                                                                                                                              |

## scalar type

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>scalarType</i> is one of:</p> <ul style="list-style-type: none"> <li>(a) <i>standardType</i>      % int, real, boolean or string</li> <li>(b) <i>enumeratedType</i></li> <li>(c) <i>subrangeType</i></li> <li>(d) <i>pointerType</i></li> <li>(e) <b>char</b></li> <li>(f) <b>int</b><math>n</math></li> <li>(g) <b>nat</b><math>n</math></li> <li>(h) <b>real</b><math>n</math></li> <li>(i) <i>namedType</i>      % Must name one of the above types</li> </ul> |
| Description | Scalar types are sometimes called <i>simple</i> or <i>primitive</i> types. The non-scalar types are strings, sets, arrays, records, unions and in OOT <b>char</b> ( $n$ ). They are defined using scalar types. Scalar types are passed by value to parameters, while non-scalars are passed by reference (by passing an implicit pointer to the non-scalar value).                                                                                                       |
| Description | In current Turing implementations scalar types are directly represented in 1, 2, 4 or 8 bytes in a computer's memory. This implies that they can be efficiently passed by value.                                                                                                                                                                                                                                                                                          |

## seek (file) statement

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>seekStatement</i> is one of:</p> <ul style="list-style-type: none"><li>(a) <b>seek</b> : <i>fileNumber</i> , <i>filePosition</i></li><li>(b) <b>seek</b> : <i>fileNumber</i> , *</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description | <p>Random access of both source (ASCII or EBCDIC) and internal form (binary) files is provided by the <b>seek</b> and <b>tell</b> statements. The <b>seek</b> statement repositions the specified file so that the next input/output operation will begin at the specified point (<i>filePosition</i>) in the file.</p> <p>The <i>fileNumber</i> must specify a file that is open with <b>seek</b> capability. The <i>filePosition</i> is a non-negative integer offset in bytes from the beginning of the file. Usually, this is a number returned by the <b>tell</b> statement. (The first position in the file is position zero.)</p> <p>Form (b) specifies that the next operation is to begin at the position immediately following the current end of the file. A <i>filePosition</i> of zero specifies that the next operation is to start at the beginning of the file. Seeking to a position beyond the current end of the file and then writing, automatically fills the intervening positions with the internal representation of zero.</p> |
| Example     | <p>This example shows how to use <b>seek</b> to append to the end of a file.</p> <pre>var employeeRecord :<br/>  record<br/>    name : string ( 30 )<br/>    pay : int<br/>  end record<br/>var fileNo : int<br/>open : fileNo, "payroll", write, seek, mod<br/>seek : fileNo, *           % Seek to the end of the file<br/>write : fileNo, employeeRecord<br/>                           % This record is added to the end of the file</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| See also    | <p><b>read</b>, <b>write</b>, <b>open</b>, <b>close</b>, <b>tell</b>, <b>get</b> and <b>put</b> statements. Another example use of <b>seek</b> is given with the explanation of the <b>tell</b> statement.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

## **self**      pointer to current object

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>self</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Description | The <b>self</b> function produces a pointer to the current object. This function can be used only inside a class declaration. See <b>class</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Example     | <p>Enter the current object onto a list of displayable objects. The module called <i>displayable</i> has exported a procedure called <i>enter</i> whose parameter type is <b>pointer to anyclass</b>. Since <b>self</b> is a pointer to <i>C</i> and <i>C</i> is a descendant of <b>anyclass</b>, it is legal to pass <b>self</b> to <i>displayable.enter</i>.</p> <pre>class C   import displayable   ...   displayable.enter ( self ) ...   ... end C</pre>                                                                                                                           |
| Details     | <p>It is illegal to call the exported entries of a class until the current object has been completely initialized, so, many calls to the current object using <b>self</b> will not be legal.</p> <p>The notation to call exported subprogram <i>p</i> of an enclosing class <i>C</i> or of its ancestor <i>D</i>, is <i>C.p</i> or <i>D.p</i>. Calls of this form, which can appear only within class <i>C</i>, call the subprogram in <i>C</i> (or in <i>D</i> in the case of <i>D.p</i>) regardless of the object type, or of any overriding, or of the status of initialization.</p> |

## **separator**      between tokens in a program

|             |                                                                                                                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | A Turing program is made up of a sequence of <i>tokens</i> (see <i>tokens</i> ), such as <b>var</b> , <i>x</i> , <b>:</b> , and <b>int</b> . These tokens may have <i>separators</i> between them. A separator is a comment (see <i>comment</i> ), blank, tab, form feed or an end of line. |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

# serialget serial port function



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>serialget : int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Description | The <b>serialget</b> procedure is used on a PC to read a single character from the serial port. This port corresponds to the MS-DOS device "COM1". This function can be used to send characters to a modem or to another computer through a null-modem cable.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Example     | <p>This program sends a message to the COM1 port and then gets a reply ending with a newline.</p> <pre>% We assume that the serial port has been set up previously using % the DOS <b>mode</b> command. <b>const</b> message : <b>string</b> := "This is a test" <b>for</b> i : 1.. <b>length</b> (message)     <b>serialput</b> (<b>ord</b> (message (i))) <b>end for</b> <b>serialput</b> (10) <b>var</b> inmessage : <b>string</b> := "" <b>var</b> ch : <b>int</b> <b>loop</b>     ch := <b>serialget</b>     <b>exit when</b> ch = 10     inmessage := inmessage + <b>chr</b> (ch) <b>end loop</b>  <b>put</b> inmessage</pre>                                                                   |
| Details     | <p><b>serialput</b> and <b>serialget</b> assume that the port's baud rate, stop bits, etc. have been properly set with the DOS <b>mode</b> command outside of Turing. Check the DOS manual for information on how to use this command.</p> <p><b>serialput</b> and <b>serialget</b> will work reliably for communication up to 1200 baud. For rates higher than this, these routines are not suitable.</p> <p>Note that these routine pass integers that represent characters. Consequently, they should be passed (and will return) values that are between 0 and 255. Because they pass integers, use the <b>chr</b> and <b>ord</b> functions to convert integers to characters and vice-versa.</p> |
| See also    | <b>serialput</b> procedure which sends characters out the serial port. See also <b>chr</b> and <b>ord</b> functions for converting between a character and its ASCII value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

# serialput serial port procedure



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>serialput</b> ( <i>p</i> : int )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Description | The <b>serialput</b> procedure is used on a PC to send a single character out the serial port. This port corresponds to the MS-DOS device "COM1". This function can be used to send characters to a modem or to another computer through a null-modem cable.                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Example     | <p>This program sends a message to the COM1 port and then gets a reply ending with a newline.</p> <pre>% We assume that the serial port has been set up previously using % the DOS <b>mode</b> command. <b>const</b> message : <b>string</b> := "This is a test " <b>for</b> i : 1.. <b>length</b> (message)     <b>serialput</b> (ord (message (i))) <b>end for</b> <b>serialput</b> (10) <b>var</b> inmessage : <b>string</b> := "" <b>var</b> ch : <b>int</b> <b>loop</b>     ch := <b>serialget</b>     <b>exit when</b> ch = 10     inmessage := inmessage + <b>chr</b> (ch) <b>end loop</b>  <b>put</b> inmessage</pre>                                                                |
| Details     | <p><b>serialput</b> and <b>serialget</b> assume that the port's baud rate, stop bits, etc. have been set with the DOS <b>mode</b> command outside of Turing. Check the DOS manual for information on how to use this command.</p> <p><b>serialput</b> and <b>serialget</b> will work reliably for communication up to 1200 baud. For rates higher than this, these routines are not suitable.</p> <p>Note that these routine pass integers that represent characters. Consequently, they should be passed (and will return) values that are between 0 and 255. Because they pass integers, use the <b>chr</b> and <b>ord</b> functions to convert integers to characters and vice-versa.</p> |
| See also    | <b>serialget</b> procedure which reads characters from the serial port. See also <b>chr</b> and <b>ord</b> functions for converting between a character and its ASCII value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

# set type

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>setType</i> is:</p> <p style="text-align: center;"><b>set of</b> <i>typeSpec</i></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Description | <p>Each value of a <b>set</b> type consists of a set of elements. The <i>typeSpec</i>, which is restricted to being a subrange or an enumerated type, gives the type of these elements.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Example     | <p>The <i>smallSet</i> type is declared so that it can contain any and all of the values 0, 1 and 2. Variable <i>s</i> is initialized to be the set containing 1 and 2.</p> <pre><b>type</b> smallSet : <b>set of</b> 0 .. 2 <b>var</b> s : smallSet := smallSet ( 0, 1 ) ... <b>if</b> 2 <b>in</b> s <b>then</b> ...</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Details     | <p>In classical mathematics, the set consisting of 0 and 1 is written as {0,1}. This is written in Turing using a <i>set constructor</i> consisting of the name of the set type followed by a parenthesized list of elements, which in this example is <i>smallInt</i> (0,1). The empty set is written, for example, as <i>smallInt</i> (). The full set is written as <i>smallInt</i> (<b>all</b>), so <i>smallInt</i> (<b>all</b>) = <i>smallInt</i> (0,1,2).</p> <p>Sets can be assigned as a whole (to sets of an equivalent type). See also <i>equivalence of types</i>.</p> <p>The operators to combine two sets are union (+), intersection (*), set subtraction (-), equality (=), inequality (<b>not=</b>), subset (&lt;=), strict subset (&lt;), superset (&gt;=), strict superset (&gt;), and <b>xor</b> ("exclusive or" also known as symmetric difference). Only sets with equivalent types (equal bounds on their index types) can be combined by these operators. The operators which determine if an element is, or is not, in a set are <b>in</b> and <b>not in</b>. For example, the test to see if 2 is in set <i>s</i> is written in the above example as: 2 <b>in</b> <i>s</i>.</p> <p>The <i>indexType</i> of a <b>set</b> type must contain at least one element. For example, the range 1 .. 0 would not be allowed. See also <i>indexType</i>. In Turing, sets are limited to at most 31 elements. OOT allows a very large number of elements.</p> |
| Details     | <p>It is illegal to declare an "anonymous" set. The only legal declaration for an <b>set</b> is in a type declaration. For example, the following is now illegal:</p> <pre><b>var</b> a : <b>array</b> 1 .. 10 <b>of set of</b> 0 .. 3</pre> <p>Given that there is no (easy) way of generating a set value without it being a named type, this should not impact any but the most bizarre code.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| See also    | <p><i>precedence</i> of operators for the order of applying set operations.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

# setConstructor

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>setConstructor</i> is:</p> $\text{setTypeId} ( \text{membersOfSet} )$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Description | <p>Each value of a <b>set</b> type consists of a set of elements. In classical mathematics, the set consisting of 0 and 1 is written as {0,1}. This is written in Turing using a <i>set constructor</i> consisting of the name of the set type (<i>setId</i>) followed by a parenthesized list of elements.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Example     | <p>The <i>smallSet</i> type is declared so that it can contain any and all of the values 0, 1 and 2. Variable <i>s</i> is initialized to be the set containing 1 and 2. The set {0,1} is written in this Turing example as <i>smallInt</i> (0,1).</p> <pre>type smallSet : set of 0 .. 2 var s : smallSet := smallSet ( 0, 1 ) ...       if 2 in s then ...</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Details     | <p>The form of <i>membersOfSet</i> is one of:</p> <ul style="list-style-type: none"><li>(a) <i>expn</i> { , <i>expn</i> } % List of members of set</li><li>(b) <b>all</b> % All member of index type of set</li><li>(c) % Nothing, meaning the empty set</li></ul> <p>The empty set is written, for example, as <i>smallInt</i> (). The full set is written as <i>smallInt</i> (<b>all</b>), so <i>smallInt</i> (<b>all</b>) = <i>smallInt</i> (0,1,2). See also the <b>set</b> type.</p> <p>The syntax of <i>setConstructor</i> as given above has been simplified by ignoring the fact that <b>set</b> types can be exported from modules. When a <b>set</b> type is exported and used outside of a module, you must write the module name, a dot and then the type name. For example, the set constructor above would be written as <i>m.smallSet</i>(1,2), where <i>m</i> is the module name.</p> |

## setpriority procedure

|        |                                       |
|--------|---------------------------------------|
| Syntax | <b>setpriority</b> ( <i>p</i> : nat ) |
|--------|---------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | The <b>setpriority</b> procedure is used to set the priority of a process in a concurrent program. This priority cannot be counted on to guarantee critical access to shared variables. A smaller value of <i>p</i> means increased speed. The argument to <b>setpriority</b> may be limited to the range 0 to 2**15 - 1. |
| See also    | <b>getpriority</b> , <b>fork</b> and <b>monitor</b> .<br>See also predefined unit <b>Concurrency</b> .                                                                                                                                                                                                                    |

## setscreen graphics procedure

Syntax            **setscreen** ( *s* : **string** )

Example           Here are example uses of the **setscreen** procedure. In many cases, these will appear as the first statement of the program. They can, however, appear any place in a program.

```

setscreen ("graphics") % IBM CGA graphics
setscreen ("graphics:e16") % IBM EGA graphics
setscreen ("screen") % To use locate
setscreen ("nocursor") % Turn off cursor
setscreen ("noecho") % Do not echo keys

```

Description      The **setscreen** statement is used to change the mode of the screen, as well as the way in which Turing does input and output. The parameter to **setscreen** is a string, such as "graphics". The string contains one or more options separated by commas, such as "text, noecho".

Details            Users should look at **View.Set** in order to find out the implementation specified details of **setscreen** on their systems.

Many of the options to **setscreen** are specific to IBM PC compatible computers and Apple Macintoshes. They have been adopted to other systems such as the Icon and X Windows.

There are three screen modes, **text**, **screen** and **graphics**. Most systems do not support all three modes.

| <u>System</u>    | <u>text</u> | <u>screen</u> | <u>graphics</u> |                    |
|------------------|-------------|---------------|-----------------|--------------------|
| <b>PC Turing</b> |             | X             | X               | (Default: screen)  |
| <b>MacTuring</b> | X           |               | X               | (Default: 480x275) |

**text** mode does not allow for any character graphics whatsoever. Only **put** and **get** are allowed.

**screen** mode allows for character graphics commands such as **locate** and **lolor**.



**graphics** mode allows for character graphics and pixel graphics commands such as **locate** and **drawbox**.

If a character graphics command is given while in **text** mode, the program automatically switches to **screen** mode if available, otherwise it switches to **graphics** mode. If a pixel graphics command is given while in **text** or **screen** mode, the program automatically switches to **graphics** mode.

Where the options to **setscreen** are mutually exclusive, they are listed here with the default underlined. Here are the options:

**text**, **screen**, **graphics** - Sets mode to the given mode and always erases the screen, even when already in the requested mode; *text* is the default character output mode; *screen* is character graphics mode, allowing the **locate** procedure; *graphics* is *pixel graphics* mode.

By default, *graphics* is CGA graphics on IBM PC compatibles. On UNIX dumb terminals, *graphics* is meaningless. A suffix can be given, as in *graphics:h16*, to specify another version of pixel graphics (assuming the corresponding hardware is available).

The set of *graphics* options is given on the next page. On Apple Macintoshes, the graphics screen is 480x275 by default. The size of the screen can also be set with a suffix in the form *graphics:150;250*, which would set the graphics output window to be 150x250 pixels. Macintoshes also allow you to set the window size in *screen* mode. The default is 25 rows by 80 columns but this can be changed with *screen:10;100* to be 10 rows by 100 columns.

Under MacTuring, **graphics** mode can have a modifier in the form *graphics:<x>;<y>*. This sets the window to be <x> by <y> in size. The maximum size of a window is the size of the screen. Also available under MacTuring, **graphics** mode can have a modifier in the form *graphics:<x>;<y>;<depth>*. This sets the window to be <x> by <y> in size. *<depth>* has the following allowable values:

**bw** or **1** or **2** = allows 2 colors (black and white)

**4** = allows 4 colors

**8** or **256** = allows 256 colors

**16** = allows 16 colors

**thousands** = 16 bit color (thousands of colors)

**millions** = 32 bit color (millions of colors)

**cga** = The window will use the 4 color CGA palette.

**ega** or **vga** = The window will use the 16 color VGA palette.

**mcpa** = The window will use the 256 color MCPA palette.

Note that the depths that use IBM color palettes have a window with a black background. They also use the IBM characters (i.e. character sizes are 8x8, 8x14 or 8x16 as they are on the IBM PC).

The set of *graphics* options is given on the next page. On Apple Macintoshes, the graphics screen is 480x275 by default. The size of the screen can also be set with a suffix in the form *graphics:150;250*, which sets the graphics output window to 150x250 pixels. Macintoshes also allow you to set the window size in *screen* mode. The default is 25 rows by 80 columns but this can be changed with *screen:10;100* to be 10 rows by 100 columns.

|                  | <u>Mode</u> |     | <u>maxx+1</u> | <u>maxy+1</u> | <u>maxcolor+1</u> |
|------------------|-------------|-----|---------------|---------------|-------------------|
| "graphics"       | 320         | 200 | 4             |               |                   |
| "graphics:cga"   | 320         | 200 | 4             |               |                   |
| "graphics:ega"   | 640         | 350 | 16            |               |                   |
| "graphics:e16"   | 640         | 350 | 16            |               |                   |
| "graphics:vga"   | 640         | 480 |               | 16            |                   |
| "graphics:v16"   | 640         | 480 | 16            |               |                   |
| "graphics:mcga"  | 320         | 200 | 256           |               |                   |
| "graphics:m256"  | 320         | 200 | 256           |               |                   |
| "graphics:svga"  | 640         | 480 | 256 (DOS OOT) |               |                   |
| "graphics:svga1" | 640         | 400 | 256 (DOS OOT) |               |                   |
| "graphics:svga2" | 640         | 480 | 256 (DOS OOT) |               |                   |
| "graphics:svga3" | 800         | 600 | 16 (DOS OOT)  |               |                   |
| "graphics:svga4" | 800         | 600 | 256 (DOS OOT) |               |                   |
| "graphics:svga5" | 1024        | 768 | 16 (DOS OOT)  |               |                   |
| "graphics:svga6" | 1024        | 768 | 256 (DOS OOT) |               |                   |

**"cursor"**, **"nocursor"** - Causes the cursor to be shown (or hidden). There is never a cursor showing in "graphics" mode. On UNIX dumb terminals, the cursor cannot be hidden. There is an optional suffix for "cursor" that determines the shape of the cursor. In CGA graphics, the cursor is constructed out of horizontal lines numbered 0, 1, 2, up to 7, with 0 being the top. The suffix gives the range of lines to be used for the cursor, for example, "cursor:5;7" specifies a cursor consisting of lines 5 through 7. In general, this form is "cursor:startline;endline", where startline and endline are integer literals such as 5 and 7. On the Apple Macintosh, it is possible to set the cursor size from 0-10.

**"echo"**, **"noecho"** - Causes (or suppresses) echoing of characters that are typed. Echoing is commonly turned off in interactive programs to keep typed characters from being echoed at inappropriate places on the screen.

Details      The **setscreen** procedure supports all the features of the older **screen** procedure. The **screen** procedure is no longer supported, you must use **setscreen** instead.

The ability of an IBM system to support a graphics mode depends on the graphics hardware available. If the system does not seem to be supporting a particular graphics mode, use the **scrntest.exe** program provided with PC Turing and DOS OOT to test the capabilities of your graphics adapter (see chapter 3 for more details). If the graphics adapter is unrecognized by Turing or DOS OOT, but supports the VESA SuperVGA graphics standard, you can use the **-vesa** startup option with Turing or DOS OOT to force the use of the VESA standard.

See also      **drawdot**, **drawline**, **drawoval**, **drawarc**, **whatdotcolor**, **color**, **colorback**, **takepic** and **drawpic**.  
See also predefined unit **View**.

## shl shift left operator

|             |                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | $A \text{ shl } B$                                                                                                                                                                                                                                                                                                                                                                                                       |
| Description | The <b>shl</b> (shift left) operator produces the value of $A$ shifted $B$ bits to the left. Both $A$ and $B$ must be non-negative integers (natural numbers).                                                                                                                                                                                                                                                           |
| Example     | Assign the base 2 value 11 to $i$ and then shift it left by 2 places and assign the resulting base 2 value 1100 to $j$ .<br><pre>var i, j : int i := 2 # 11      % 2#11 = 3 (base 10) j := i shl 2      % j becomes 2#1100 = 12 (base 10)</pre>                                                                                                                                                                          |
| Details     | The <b>shl</b> operator is defined mathematically (in a machine-independent way) as follows: $A \text{ shl } B = A * (2^{**}B)$ . Overflow occurs when the result exceeds the maximum value of the <b>nat4</b> (4-byte natural number) type.<br>Value $A$ can be of any integer type (as long as it is non-negative) or any natural number type.<br>The <b>shl</b> operator has the same precedence as the $*$ operator. |
| See also    | <b>shr</b> (shift right), <b>or</b> , <b>and</b> and <b>xor</b> , which also are bit manipulation operators that act on non-negative values. See also <b>explicitIntegerConstant</b> which describes values such as 2#1100.                                                                                                                                                                                              |

## shr shift right operator

|             |                                                                                                                                                                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | $A \text{ shr } B$                                                                                                                                                                                                                               |
| Description | The <b>shr</b> (shift right) operator produces the value of $A$ shifted $B$ bits to the right. Both $A$ and $B$ must be non-negative integers (natural numbers).                                                                                 |
| Example     | Assign the base 2 value 1101 to $i$ and then shift it right by 2 places and assign the resulting base 2 value 11 to $j$ .<br><pre>var i, j : int i := 2 # 1101    % 2#1101 = 13 (base 10) j := i shr 2      % j becomes 2#11 = 3 (base 10)</pre> |

|          |                                                                                                                                                                                                                                                                                                                                                                        |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | <p>The <b>shr</b> operator is defined mathematically (in a machine- independent way) as follows: <math>A \text{ shr } B = A \text{ div } 2^{**}B</math>.</p> <p>Value <math>A</math> can be of any integer type (as long as it is non-negative) or any natural number type.</p> <p>The <b>shr</b> operator has the same precedence as the <math>*</math> operator.</p> |
| See also | <b>shl</b> (shift left), <b>or</b> , <b>and</b> and <b>xor</b> , which also are bit manipulation operators that act on non-negative values. See also <b>explicitIntegerConstant</b> which describes values such as 2#1101.                                                                                                                                             |

## sign function

|             |                                                                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>sign</b> ( $r : \text{real}$ ) : -1 .. 1                                                                                                                                                                                           |
| Description | The <b>sign</b> function is used to determine whether a number is positive, zero or negative. It returns 1 if $r > 0$ , 0 if $r = 0$ , and -1 if $r < 0$ . For example, <b>sign</b> (5) is 1 and <b>sign</b> (-23) is -1.             |
| Example     | <p>This program reads in numbers and determines if they are positive, zero or negative:</p> <pre> var x : real get x case sign ( x ) of   label 1 : put "Positive"   label 0 : put "Zero"   label -1 : put "Negative" end case </pre> |
| See also    | See also predefined unit <b>Math</b> .                                                                                                                                                                                                |

## signal wake up a process statement

|        |                                                                                   |
|--------|-----------------------------------------------------------------------------------|
| Syntax | <p>A <i>signalStatement</i> is:</p> <p><b>signal</b> <i>variableReference</i></p> |
|--------|-----------------------------------------------------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | A <b>signal</b> statement is used in a concurrent program to wake up a process that is blocked (waiting on a condition variable). The statement can only be used inside a monitor (a special kind of module that handles concurrency). A <b>signal</b> statement operates on a <b>condition</b> variable (the <i>variableReference</i> ), which is essentially a queue of sleeping processes. See <b>condition</b> for an example of a <b>signal</b> statement.                                                                                                          |
| Details     | A <b>signal</b> statement wakes up one process that is doing a <b>wait</b> on the specified condition queue, if such a process exists. If the condition is deferred (or <b>timeout</b> ; see <b>condition</b> ), the signaler continues in the monitor, and the awakened process is allowed to continue only when the monitor becomes inactive. A signal to an <i>immediate</i> (non-deferred) condition causes the signaled process to begin running in the monitor immediately. The signaling process waits to re-enter the monitor when the monitor becomes inactive. |
| See also    | <b>condition</b> and <b>wait</b> . See also <b>monitor</b> and <b>fork</b> . See also <b>empty</b> . See also <b>pause</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                             |

## simutime      simulated time function

|             |                                                                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>simutime : int</b>                                                                                                                                                                                                                                |
| Description | The <b>simutime</b> function returns the number of simulated time units that have passed since program execution began.                                                                                                                              |
| Details     | Simulated time only passes when all process are either paused or waiting. This simulates the fact that CPU time is effectively infinitely fast when compared to "pause" time.                                                                        |
| Example     | <p>This prints out the simulated time passing between two processes. This will print out 3, 5, 6, 9, 10, 12, 15, 15, 18, 20, 21, ...</p> <pre> process p (t : int)   loop     pause t     put simutime   end loop end p  fork p (3) fork p (5)</pre> |
| See also    | See also predefined unit <b>Concurrency</b> .                                                                                                                                                                                                        |

## **sin**    sine function (radians)

Syntax            **sin ( *r* : real ) : real**

Description      The **sin** function is used to find the sine of an angle given in radians. For example, **sin** (0) is 0.

Example          This program prints out the sine of  $\pi/6$ ,  $2\pi/6$ ,  $3\pi/6$ , up to  $12\pi/6$  radians.

```
const pi := 3.14159
for i : 1 .. 12
 const angle := i * pi / 6
 put "Sin of", angle, " is ", sin (angle)
end for
```

See also          the **sind** function which finds the sine of an angle given in degrees. ( $2\pi$  radians are the same as 360 degrees.)  
See also predefined unit **Math**.

## **sind**    sine function (degrees)

Syntax            **sind ( *r* : real ) : real**

Description      The **sind** function is used to find the sine of an angle given in degrees. For example, **sind** (0) is 0.

Example          This program prints out the sine of 30, 60, 90, up to 360 degrees.

```
for i : 1 .. 12
 const angle := i * 30
 put "Sin of", angle, " is ", sind (angle)
end for
```

See also          the **sin** function which finds the sine of an angle given in radians. ( $2\pi$  radians are the same as 360 degrees.)  
See also predefined unit **Math**.

## sizeof      size of a type Dirty

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>sizeof</b> ( <i>typeNameOrVariableReference</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Description | The <b>sizeof</b> attribute is used to find the number of bytes used to represent the type or variable. This is implementation-dependent (dirty).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Example     | The size of <b>int2</b> and <b>nat2</b> is 2.<br><pre>var i : int2       const nat2size := sizeof ( i )    % size is 2</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Details     | <p>The <i>typeNameOrVariableReference</i> must be the name of a user-defined type, a variable reference, a basic type (such as <b>real</b>), or a constant.</p> <p>In principle, <b>sizeof</b> returns the number of <i>storage units</i> which would not necessarily be 8-bit bytes. For example, in some older machines, such as the CDC 6000 series, the storage units are 60 bit words. However, almost all modern computers use 8-bit bytes so these are the units of <b>sizeof</b>.</p> <p>Beware that sizes may reflect alignment constraints in the underlying computer. For example, string sizes may be rounded up to even values (2-byte word alignments).</p> |
| See also    | the <i>indirection operator</i> @, <b>cheat</b> , <i>explicitIntegerConstant</i> (how to write hexadecimal constants), and pointers (in particular unchecked pointers). See also <b>addr</b> , which returns the address of a variable.                                                                                                                                                                                                                                                                                                                                                                                                                                   |

## sizepic      graphics function Pixel graphics only

|             |                                                                                                                                                                                                                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>sizepic</b> ( <i>x1, y1, x2, y2 : int</i> ) : int                                                                                                                                                                                                                                                                    |
| Description | The <b>sizepic</b> function is used to determine the size buffer needed to record a picture from the screen (see description of <b>takepic</b> ). This gives the minimum number of elements of the int array used by <b>takepic</b> . The buffer is used by <b>drawpic</b> to make copies of the picture on the screen. |
| Example     | This program outputs the size of array needed to hold a picture with left bottom corner at x=10, y=20 and right top corner at x=50, y=60.                                                                                                                                                                               |

|          |                                                                                                                                                                                                                                                                                                                                     |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          | <pre> <b>setscreen</b> ("graphics") ... <b>put</b> "The size of the array needs to be",       <b>sizepic</b> ( 10, 20, 50, 60 ) </pre>                                                                                                                                                                                              |
| Details  | <p>See <b>takepic</b> for an example of the use of <b>sizepic</b> and for further information about buffers for drawing pictures.</p> <p>The screen should be in a "graphics" mode. See the <b>setscreen</b> procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p> |
| See also | <p><b>drawpic</b>. See also <b>setscreen</b>, <b>maxx</b>, <b>maxy</b>, <b>drawdot</b>, <b>drawline</b>, <b>drawbox</b>, and <b>drawoval</b>.</p> <p>See also predefined unit <b>Pic</b>.</p>                                                                                                                                       |

## skip    used in get statement

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>skip</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Description | Using <b>skip</b> as an input item in a <b>get</b> statement causes the current input to be skipped until a non-whitespace token is encountered. Whitespace includes all blanks, tabs, form feeds and newlines.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Example     | <p>The <b>skip</b> input item was originally intended to be used to see if more input exists in an input file. This use has been largely made redundant by a change in the Turing language. The new version of Turing reads a token, as in <i>get s</i> but not in <i>get s:*</i> or <i>get s:3</i>, and automatically skips any white space following the input value, but will not go beyond the beginning of the next input line. Originally this automatic skipping did not take place, so <b>skip</b> was required. The form of an input loop that used <b>skip</b> was as follows:</p> <pre> loop   get skip           % This is line now redundant   exit when eof   get ...   ... end loop </pre> |
| Details     | <p>The <b>skip</b> bypasses all whitespace characters including any trailing newlines and blank lines. By skipping these characters, a true end-of-file condition was detected. Otherwise, the end-of-file could have been hidden by any whitespace following the last input item. With the change in Turing, the line <b>get skip</b> is no longer needed (although it still works correctly).</p>                                                                                                                                                                                                                                                                                                       |



Example      The **skip** can also be used to correctly identify the start of a long string (usually to be read in *line* or *counted* mode). Here, it skips the whitespace and trailing newline as follows:

```
var i : int
var line : string
loop
 get i, skip, line:*
 ...
end loop
```

Details      The first item in the **get** statement reads an integer by skipping all whitespace and reading digits until whitespace is encountered. The input stream is then left with the whitespace as the next input character. The **skip** then skips past the whitespace, effectively beginning the next input at the next non-whitespace character. This truncates leading blanks and has another, potentially more important, effect. If the integer is the last data on a line and the string is on a following line, the **skip** is necessary to avoid setting *line* to a null string value.

See also      **get** statement and **loop** statement.

## skip      used in put statement

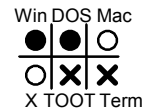
Syntax      **skip**

Description      Using **skip** as an output item in a **put** statement causes the current output line to be ended and a new line to be started.

Example      This example, *To be* is output on one line and *Or not to be* on the next.  
                  **put** "To be", **skip**, "Or not to be"

Details      Using **skip** is equivalent to outputting the newline character "\n".


## sound      statement



Syntax      **sound** ( *frequency*, *duration* : **int** )

|             |                                                                                                                                                                                                                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | The <b>sound</b> procedure is used to cause the computer to sound a note of a given frequency for a given time. The frequency is in cycles per second (Hertz). The time duration is in milliseconds. For example, middle A on a piano is 440 Hertz, so <b>sound</b> (440, 1000) plays middle A for one second. |
| Example     | <p>This program sounds the frequencies 100, 200 up to 1000 each for half a second.</p> <pre> <b>for</b> i : 1 .. 10   <b>put</b> i   <b>sound</b> ( 100 * i, 500 )  % Sound note for 1/2 second <b>end for</b> </pre>                                                                                          |
| Details     | On IBM PC compatibles, the hardware resolution of duration is in units of 55 milliseconds. For example, <b>sound</b> (440, 500) will delay the program by about half a second, but may be off by as much as 55 milliseconds.                                                                                   |
| Details     | The <b>sound</b> procedure does not currently work under MacOOT.                                                                                                                                                                                                                                               |
| See also    | <b>play</b> statement, which plays notes based on musical notation. For example, <b>play</b> ("8C") plays an eighth note of middle C. See also the <b>delay</b> , <b>clock</b> , <b>sysclock</b> , <b>wallclock</b> , <b>time</b> and <b>date</b> statements.<br>See also predefined unit <b>Music</b> .       |

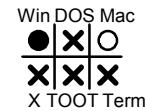
## Sprite

Win DOS Mac  
  
 X TOOT Term

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                          |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description  | <p>Sprites are a way of doing animation in Turing bypassing the <b>Pic</b> module. A sprite is essentially a picture with a specific location and "depth". You create a sprite by calling <b>Sprite.New</b> with a picID received from <b>Pic.New</b>. You can then move the sprite around by calling <b>Sprite.SetPosition</b>. When you are finished with the sprite, you call <b>Sprite.Free</b>.</p> <p>Note that sprites work best when they are moderately small. If you have large sprites, you will continue to have flashing.</p> <p>All subprograms in the <b>Sprite</b> unit are exported qualified (and thus must be prefaced with "<b>Sprite.</b>").</p> |                                                                                                                                                                          |
| Entry Points | <b>New</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Creates a new sprite from a picture.                                                                                                                                     |
|              | <b>Free</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Disposes of a sprite and free up its memory.                                                                                                                             |
|              | <b>SetHeight</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Sets the height of a sprite. Sprites with a greater height appear above sprites with a lesser height. The background is considered height 0. The height may be negative. |

|                    |                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------|
| <b>SetPosition</b> | Sets the location of the sprite. Can specify the center of the sprite or the lower-left corner.            |
| <b>ChangePic</b>   | Changes the picture associated with a sprite.                                                              |
| <b>Animate</b>     | Changes the location and the picture associated with a sprite. Used for animating a moving changing image. |
| <b>Show</b>        | Shows a previously hidden sprite.                                                                          |
| <b>Hide</b>        | Hides a visible sprite.                                                                                    |

## Sprite.Animate

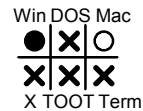


|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sprite.Animate</b> ( <i>spriteID</i> , <i>picID</i> , <i>x</i> , <i>y</i> : <b>int</b> ,<br><i>centered</i> : <b>int</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Description | <p>Moves the sprite specified by <i>spriteID</i> to the location specified by (<i>x</i>, <i>y</i>). If <i>centered</i> is <b>true</b>, then the sprite is centered on (<i>x</i>, <i>y</i>). Otherwise (<i>x</i>, <i>y</i>) specifies the lower-left corner of the sprite. At the same time, it changes the picture associated with the sprite.</p> <p>A simple example of the <b>Sprite.Animate</b> procedure would be of a man walking. The picture associated with the sprite would constantly change as the figure was walking. At the same time, the location of the figure would also change.</p>                                   |
| Example     | <p>Here is a program that loads six images from files <i>Pic1.bmp</i> through <i>Pic6.bmp</i> and then displays them sequentially on the screen, moving the image two pixels each time.</p> <pre> var pics : array 0 .. 5 of int var sprite: int for i : 1 .. 6   pics (i - 1) := Pic.FileNew ("Pic" + intstr (i) + ".bmp")   if Error.Last not= 0 then     put "Error loading image: ", Error.LastMsg     return   end if end for sprite:= Sprite.New (pics (0)) Sprite.SetPosition (sprite, 0, 100, false) Sprite.Show (sprite) for x : 2 .. maxx by 2   Sprite.Animate (sprite, pics ((x div 2) mod 6), x, 100, false) end for </pre> |

## **Sprite.Free** (*sprite*)

|          |                                                                                                                                          |
|----------|------------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Sprite.Animate</b> , not by calling <b>Animate</b> . |
| See also | <b>Sprite.New</b> , <b>Sprite.SetPosition</b> and <b>Sprite.ChangePic</b> .                                                              |

# Sprite.ChangePic



|             |                                                                                                                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sprite.ChangePic</b> ( <i>spriteID</i> , <i>picID</i> : <b>int</b> )                                                                                                                             |
| Description | Changes the picture associated with a sprite while maintaining the sprites height and visibility status. A typical use <b>Sprite.ChangePic</b> would be to animate a sprite that stays in position. |
| Example     | Here is a program that t that loads six images from files <i>Pic1.bmp</i> through <i>Pic6.bmp</i> and then displays them sequentially in the center of the screen.                                  |

```
var pics : array 0 .. 5 of int
var sprite: int
for i : 1 .. 6
 pics (i - 1) := Pic.FileNew ("Pic" + intstr (i) + ".bmp")
 if Error.Last not= 0 then
 put "Error loading image: ", Error.LastMsg
 return
 end if
end for
figure := Sprite.New (pics (0))
Sprite.SetPosition (sprite, maxx div 2, maxy div 2, true)
Sprite.Show (sprite)
for i : 1 .. 100
 Sprite.ChangePic (sprite, pics (i mod 6))
end for
Sprite.Free (sprite)
```

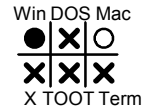
|          |                                                                                                                                              |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Sprite.ChangePic</b> , not by calling <b>ChangePic</b> . |
| See also | <b>Sprite.New</b> .                                                                                                                          |

# Sprite.Free



|             |                                                                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sprite.Free</b> ( <i>spriteID</i> : int )                                                                                            |
| Description | Destroys the sprite and frees up the memory the sprite used. It is an error to use the <i>spriteID</i> after the sprite has been freed. |
| Example     | See <b>Sprite.Animate</b> for an example of <b>Sprite.Free</b> .                                                                        |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Sprite.Free</b> , not by calling <b>Free</b> .      |
| See also    | <b>Sprite.New</b> .                                                                                                                     |

# Sprite.Hide



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sprite.Hide</b> ( <i>spriteID</i> : int )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Description | Hides a previously visible sprite. <b>Sprite.Hide</b> has no effect if the sprite is already invisible.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Example     | <p>The following program animates four balls on the screen. When the balls are close to each other or to a wall, they appear, otherwise they are hidden.</p> <pre> var pic, sprite, x, y, dx, dy, radius : array 1 .. 6 of int var visible : array 1 .. 6 of boolean  setscreen ("nocursor")  % Create all the sprites. for i : 1 .. 6     radius (i) := Rand.Int (10, 25)     Draw.FillOval (25, 25, radius (i), radius (i), 8 + i)     Font.Draw (intstr (i), 20, 20, 0, black)     pic (i) := Pic.New (0, 0, 50, 50)     Draw.FillBox (0, 0, 50, 50, 0)     x (i) := Rand.Int (radius (i), maxx &lt; radius (i))     y (i) := Rand.Int (radius (i), maxy &lt; radius (i)) </pre> |

```

 dx (i) := Rand.Int (3, 3)
 dy (i) := Rand.Int (3, 3)
 sprite (i) := Sprite.New (pic (i))
 Sprite.SetPosition (sprite (i), x (i), y (i), true)
 Sprite.SetHeight (sprite (i), i)
 visible (i) := false
end for

% Now move all the sprites around the screen.
loop
 for i : 1 .. 6
 if x (i) + dx (i) < radius (i) or
 x (i) + dx (i) > maxx - radius (i) then
 dx (i) := -dx (i)
 end if
 x (i) := x (i) + dx (i)
 if y (i) + dy (i) < radius (i) or
 y (i) + dy (i) > maxy - radius (i) then
 dy (i) := -dy (i)
 end if
 y (i) := y (i) + dy (i)
 end for
 for i : 1 .. 6
 var near : boolean := false
 if (x (i) < 50) or (x (i) > maxx - 50) or
 (y (i) < 50) or (y (i) > maxy - 50) then
 near := true
 end if
 if not near then
 for j : 1 .. 6
 if i not= j then
 if sqrt ((x (i) - x (j)) ** 2 +
 (y (i) - y (j)) ** 2) < 100 then
 near := true
 exit
 end if
 end if
 end for
 end if
 if near and not visible (i) then
 Sprite.Show (sprite (i))
 visible (i) := true
 elsif not near and visible (i) then
 Sprite.Hide (sprite (i))
 visible (i) := false
 end if
 Sprite.SetPosition (sprite (i), x (i), y (i), true)
 end for
 Time.Delay (40)
 exit when hasch
end loop
for i : 1 .. 6
 Sprite.Free (sprite (i))
end for

```

Status                      Exported qualified.

This means that you can only call the function by calling **Sprite.Hide**, not by calling **Hide**.

See also **Sprite.Show**.

## Sprite.New

Win DOS Mac  
● X O  
X X X  
X TOOT Term

Syntax **Sprite.New** ( *picID* : **int** ) : **int**

Description Creates a new sprite from a picture specified by *picID*. The sprite starts invisible and should be given a *depth* using **Sprite.SetHeight** and a position, given **Sprite.SetPosition** before being made visible using **Sprite.Show**. When you are finished using the sprite, the sprite should be freed using **Sprite.Free**.

Sprites work best when they are of moderate size. Large sprites will cause flashing when moved across the screen.

Anything that is color 0 in the picture will not appear when the sprite is drawn. In other words, color 0 is transparent.

Example See **Sprite.Animate** for an example of **Sprite.New**.

Status Exported qualified.  
This means that you can only call the function by calling **Sprite.New**, not by calling **New**.

See also **Sprite.SetHeight**, **Sprite.SetPosition**, **Sprite.Show** and **Sprite.Free**.

## Sprite.SetHeight

Win DOS Mac  
● X O  
X X X  
X TOOT Term

Syntax **Sprite.SetHeight** ( *spriteID*, *newHeight* : **int** )

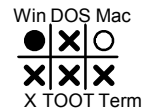
Description Sets the height of the sprite specified by *spriteID* to the value specified by *newHeight*.

The height of a sprite determines which sprite appears above another when they overlap. The "higher" sprite (the one with the greater height) will appear on top of the sprite with the lower height, even if the lower sprite is drawn second.

The background (i.e. any non-sprite) is considered to be in height 0. Sprites with a negative height will appear "behind" the background. Note that if two sprites have the same height, the one drawn last will appear above the first one.

|          |                                                                                                                                                  |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Example  | See <b>Sprite.Hide</b> for an example of <b>Sprite.SetHeight</b> .                                                                               |
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Sprite.SetPosition</b> , not by calling <b>SetPosition</b> . |
| See also | <b>Sprite.New</b> .                                                                                                                              |

## Sprite.SetPosition



|             |                                                                                                                                                                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sprite.SetPosition</b> ( <i>spriteID</i> , <i>x</i> , <i>y</i> : <b>int</b> ,<br><i>centered</i> : <b>boolean</b> )                                                                                                                                                           |
| Description | Moves the sprite specified by <i>spriteID</i> to the location specified by ( <i>x</i> , <i>y</i> ). If <i>centered</i> is <b>true</b> , then the sprite is centered on ( <i>x</i> , <i>y</i> ). Otherwise ( <i>x</i> , <i>y</i> ) specifies the lower-left corner of the sprite. |
| Example     | See <b>Sprite.Hide</b> for an example of <b>Sprite.SetPosition</b> .                                                                                                                                                                                                             |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Sprite.SetPosition</b> , not by calling <b>SetPosition</b> .                                                                                                                                 |
| See also    | <b>Sprite.New</b> .                                                                                                                                                                                                                                                              |



# Sprite.Show



|             |                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sprite.Show</b> ( <i>spriteID</i> : int )                                                                                       |
| Description | Displays a previously hidden sprite. <b>Sprite.Show</b> has no effect if the sprite is already visible.                            |
| Example     | See <b>Sprite.Hide</b> for an example of <b>Sprite.Show</b> .                                                                      |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Sprite.Show</b> , not by calling <b>Show</b> . |
| See also    | <b>Sprite.Hide</b> .                                                                                                               |

## sqrt square root function

|             |                                                                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>sqrt</b> ( <i>r</i> : real ) : real                                                                                                                                                                                                   |
| Description | The <b>sqrt</b> function is used to find the square root of a number. For example, <b>sqrt</b> (4) is 2.                                                                                                                                 |
| Example     | This program prints out the square roots of 1, 2, 3, ... up to 100.<br><pre>for i : 1 .. 100   put "Square root of ", i, " is ", sqrt ( i ) end for</pre>                                                                                |
| Details     | It is illegal to try to take the square root of a negative number. The result of <b>sqrt</b> is always positive or zero.<br>The opposite of a square root is the square. For example, the square of <i>x</i> is written is <i>x</i> **2. |
| See also    | See also predefined unit <b>Math</b> .                                                                                                                                                                                                   |

# standardType

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>standardType</i> is one of:</p> <ul style="list-style-type: none"><li>(a) <b>int</b></li><li>(b) <b>real</b></li><li>(c) <b>string</b> [ ( <i>maxLength</i> ) ]</li><li>(d) <b>boolean</b></li><li>(e) <b>nat</b> % <i>natural number</i></li><li>(f) <b>int</b><i>n</i> % <i>n-byte integer (n=1, 2, 4)</i></li><li>(g) <b>nat</b><i>n</i> % <i>n-byte natural (n= 1, 2, 4)</i></li><li>(h) <b>real</b><i>n</i> % <i>n-byte real (n=4, 8)</i></li><li>(i) <b>char</b> % <i>single character</i></li><li>(j) <b>char</b> (<i>n</i>) % <i>n characters</i></li></ul> |
| Description | The standard types can be used throughout a program. They should not be included in an <b>import</b> list.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| See also    | <b>int</b> , <b>real</b> , <b>string</b> and <b>boolean</b> . See also <b>nat</b> , <b>int</b> <i>n</i> , <b>nat</b> <i>n</i> , <b>real</b> <i>n</i> , <b>char</b> , <b>char</b> ( <i>n</i> )                                                                                                                                                                                                                                                                                                                                                                               |

# statement

|        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax | <p>A <i>statement</i> is one of:</p> <ul style="list-style-type: none"><li>(a) <i>assignmentStatement</i> % <i>variableReference := expn</i></li><li>(b) <i>openStatement</i> % <i>open ...</i></li><li>(c) <i>closeStatement</i> % <i>close ...</i></li><li>(d) <i>putStatement</i> % <i>put ...</i></li><li>(e) <i>getStatement</i> % <i>get ...</i></li><li>(f) <i>readStatement</i> % <i>read ...</i></li><li>(g) <i>writeStatement</i> % <i>write ...</i></li><li>(h) <i>seekStatement</i> % <i>seek ...</i></li></ul> |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                               |                                                                       |                     |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|---------------------|
|             | (i)                                                                                                                                                                                                                                                                                                                           | <i>tellStatement</i>                                                  | % tell ...          |
|             | (j)                                                                                                                                                                                                                                                                                                                           | <i>forStatement</i>                                                   | % for ... end for   |
|             | (k)                                                                                                                                                                                                                                                                                                                           | <i>loopStatement</i>                                                  | % loop ... end loop |
|             | (l)                                                                                                                                                                                                                                                                                                                           | <b>exit</b> [ <b>when</b> <i>trueFalseExpn</i> ]                      |                     |
|             | (m)                                                                                                                                                                                                                                                                                                                           | <i>ifStatement</i>                                                    | % if ... end if     |
|             | (n)                                                                                                                                                                                                                                                                                                                           | <i>caseStatement</i>                                                  | % case ... end case |
|             | (o)                                                                                                                                                                                                                                                                                                                           | <b>assert</b> <i>trueFalseExpn</i>                                    |                     |
|             | (p)                                                                                                                                                                                                                                                                                                                           | <b>begin</b>                                                          |                     |
|             |                                                                                                                                                                                                                                                                                                                               | <i>statementsAndDeclarations</i>                                      |                     |
|             |                                                                                                                                                                                                                                                                                                                               | <b>end</b>                                                            |                     |
|             | (q)                                                                                                                                                                                                                                                                                                                           | <i>procedureCall</i> % <i>procedureId</i> [( <i>parameters</i> )]     |                     |
|             | (r)                                                                                                                                                                                                                                                                                                                           | <b>return</b>                                                         |                     |
|             | (s)                                                                                                                                                                                                                                                                                                                           | <b>result</b> <i>expn</i>                                             |                     |
|             | (t)                                                                                                                                                                                                                                                                                                                           | <b>new</b> [ <i>collectionId</i> , ] <i>pointerVariableReference</i>  |                     |
|             | (u)                                                                                                                                                                                                                                                                                                                           | <b>free</b> [ <i>collectionId</i> , ] <i>pointerVariableReference</i> |                     |
|             | (v)                                                                                                                                                                                                                                                                                                                           | <b>tag</b> <i>unionVariableReference</i> , <i>expn</i>                |                     |
|             | (w)                                                                                                                                                                                                                                                                                                                           | <i>forkStatement</i>                                                  |                     |
|             | (x)                                                                                                                                                                                                                                                                                                                           | <b>signal</b> <i>variableReference</i>                                |                     |
|             | (y)                                                                                                                                                                                                                                                                                                                           | <b>wait</b> <i>variableReference</i> [ , <i>expn</i> ]                |                     |
|             | (z)                                                                                                                                                                                                                                                                                                                           | <b>pause</b> <i>expn</i>                                              |                     |
|             | (aa)                                                                                                                                                                                                                                                                                                                          | <b>quit</b> [ <i>guiltyParty</i> ] [ : <i>quitReason</i> ]            |                     |
| (bb)        |                                                                                                                                                                                                                                                                                                                               | <b>unchecked</b>                                                      |                     |
| (cc)        |                                                                                                                                                                                                                                                                                                                               | <b>checked</b>                                                        |                     |
| Description | <p>A <i>statement</i> (or <i>command</i>) causes a particular action, for example, the <i>putStatement</i>:</p> <pre><b>put</b> "Hello"</pre> <p>outputs <i>Hello</i>. See the descriptions of the individual statements for explanations of their actions. Each statement can optionally be followed by a semicolon (;).</p> |                                                                       |                     |
| Example     | <pre><i>width</i> := 24           % Assignment statement <b>put</b> "Hello world" % Put statement <b>exit when</b> <i>i</i> = 100    % Exit statement <b>assert</b> <i>width</i> &lt; 320 % Assert statement</pre>                                                                                                            |                                                                       |                     |
| Details     | <p>You can use a <b>result</b> statement only in a function. You can use a <b>return</b> statement only to terminate a procedure or the main program (but not to terminate the initialization of a module). See also <b>result</b> and <b>return</b>.</p>                                                                     |                                                                       |                     |

There are a number of predefined procedures, such as *drawline*, which are not listed as statements above. These are considered procedure calls, which is one form of statement.

## statementsAndDeclarations

|             |                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p><i>StatementsAndDeclarations</i> are:</p> <p style="text-align: center;"><b>{ <i>statementOrDeclaration</i> }</b></p>                                                                                                                                                                                                                                                                                      |
| Description | <p><i>StatementsAndDeclarations</i> are a list of statements and declarations. For example, a Turing program consists of a list of statements and declarations. The body of a procedure is a list of statements and declarations.</p> <p>Each <i>statementOrDeclaration</i> is one of:</p> <p>(a) <i>statement</i>                      (b) <i>declaration</i></p> <p>See also statement and declaration.</p> |
| Example     | <p>This list of statements and declarations is a Turing program that outputs <i>Hello Frank</i>.</p> <pre> <b>var</b> name : <b>string</b> name := "Frank" <b>put</b> "Hello ", name </pre>                                                                                                                                                                                                                   |

## Str All

|               |                                                                                                                                                                                                                                     |              |                                             |               |                                 |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|---------------------------------------------|---------------|---------------------------------|
| Description   | <p>This unit contains the predefined constants for manipulating strings. All routines in the Str module are exported unqualified. Descriptions of all the subprograms in the unit can be found in Chapter 4, Language Features.</p> |              |                                             |               |                                 |
| Entry Points  | <table> <tr> <td><b>index</b></td><td>Finds a specified string in another string.</td></tr> <tr> <td><b>length</b></td><td>Returns the length of a string.</td></tr> </table>                                                       | <b>index</b> | Finds a specified string in another string. | <b>length</b> | Returns the length of a string. |
| <b>index</b>  | Finds a specified string in another string.                                                                                                                                                                                         |              |                                             |               |                                 |
| <b>length</b> | Returns the length of a string.                                                                                                                                                                                                     |              |                                             |               |                                 |

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <b>repeat</b> | Creates a string by repeating a specified string a number of times. |
|---------------|---------------------------------------------------------------------|

## Stream All

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                 |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that deal with I/O streams. The basic I/O in Turing is done with I/O statements. However, extra functions are all part of the <b>Stream</b> unit.</p> <p>All routines in the <b>Stream</b> unit are exported qualified (and thus must be prefaced with "<b>Stream.</b>"), with the exception of <b>eof</b> which is part of the language but conceptually part of this unit and is considered to be exported unqualified.</p> |                                                 |
| Entry Points | <b>eof</b> *                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Determines if the end of file has been reached. |
|              | <b>Flush</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Flushes a specified stream.                     |
|              | <b>FlushAll</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Flushes all open output streams.                |
|              | * Part of the language, conceptually part of the <b>Stream</b> unit.                                                                                                                                                                                                                                                                                                                                                                                                           |                                                 |

## Stream.eof All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>eof</b> ( <i>streamNumber</i> : <b>int</b> ) : <b>boolean</b>                                                                                                                                                                                                                                                                                                                                                                   |
| Description | <p>The <b>eof</b> (end of file) function is used to determine if there is any more input. It returns <b>true</b> when there are no more characters to be read. The parameter and its parentheses are omitted when referring to the standard input (usually this is the keyboard); otherwise the parameter specifies the number of a stream. The stream number has been determined (in most cases) by an <b>open</b> statement.</p> |
| Example     | <p>This program reads and outputs all the lines in the file called "info".</p> <pre><b>var</b> line : <b>string</b></pre>                                                                                                                                                                                                                                                                                                          |

```

var fileNumber : int
open : fileNumber, "info", get
loop
 exit when eof (fileNumber)
 get : fileNumber, line : *
 put line
end loop

```

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details | <p>See also the description of the <b>get</b> statement, which gives more examples of the use of <b>eof</b>. See also the <b>open</b> and <b>read</b> statements.</p> <p>When the input is from the keyboard, the user can signal end-of-file by typing control-Z on a PC (or control-D on UNIX). If a program tests for <b>eof</b> on the keyboard, and the user has not typed control-Z (or control-D) and the user has typed no characters beyond those that have been read, the program must wait until the next character is typed. Once this character is typed, the program knows whether it is at the end of the input, and returns the corresponding <b>true</b> or <b>false</b> value for <b>eof</b>.</p> |
| Status  | <p>Part of the language and only conceptually part of the <b>Stream</b> unit.</p> <p>This means that you can only call the function by calling <b>eof</b>, not by calling <b>Stream.eof</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## Stream.Flush All

|             |                                                                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Stream.Flush</b> ( <i>streamNumber</i> : <b>int</b> )                                                                                                       |
| Description | The <b>Stream.Flush</b> procedure is used to flush any buffered output associated with the <i>streamNumber</i> parameter.                                      |
| Details     | <p>Turing automatically flushes any buffered output when a stream is closed. Turing also automatically closes any open files when execution is terminated.</p> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Stream.Flush</b>, not by calling <b>Flush</b>.</p>                  |

# Stream.FlushAll All

|             |                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Stream.FlushAll</b>                                                                                                                                  |
| Description | The <b>Stream.FlushAll</b> procedure is used to flush any buffered output in any open file.                                                             |
| Details     | Turing automatically flushes any buffered output when a stream is closed. Turing also automatically closes any open files when execution is terminated. |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Stream.FlushAll</b> , not by calling <b>FlushAll</b> .              |

## string comparison

|             |                                                                                                                                                                                                                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | A <i>stringComparison</i> is one of:<br>(a) <i>stringExpn</i> = <i>stringExpn</i><br>(b) <i>stringExpn</i> <b>not=</b> <i>stringExpn</i><br>(c) <i>stringExpn</i> > <i>stringExpn</i><br>(d) <i>stringExpn</i> < <i>stringExpn</i><br>(e) <i>stringExpn</i> >= <i>stringExpn</i><br>(f) <i>stringExpn</i> <= <i>stringExpn</i> |
| Description | Strings ( <i>stringExps</i> ) can be compared for equality (= and <b>not=</b> ) and for ordering (>, <, >= and <=).                                                                                                                                                                                                            |
| Example     | <pre>var name : string := "Nancy" var licenceNumber : string ( 6 )     licenceNumber := "175AJN"</pre>                                                                                                                                                                                                                         |

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details | <p>Two strings are considered to be equal (=) if they have the same length and are made up, character by character, of the same characters. If they differ, they are considered to be unequal (<b>not=</b>).</p> <p>Ordering among strings is essentially alphabetic order. String <i>S</i> is considered to come before string <i>T</i>, that is <math>S &lt; T</math>, if the two are identical up to a certain position and after that position, either the next character of <i>S</i> comes before the next character of <i>T</i>, or else there are no more characters in <i>S</i> while <i>T</i> contains more characters.</p> <p><math>S &gt; T</math> (<i>S</i> comes after <i>T</i>) means the same thing as <math>T &lt; S</math>. <math>S \geq T</math> means the same thing as <math>S &gt; T</math> <b>or</b> <math>S = T</math>. <math>S \leq T</math> means the same thing as <math>S &lt; T</math> <b>or</b> <math>S = T</math>.</p> <p>ASCII gives the ordering among individual characters. It specifies, among other things, that letter capital <i>L</i> comes alphabetically before capital letter <i>M</i> and similarly for small (lower case) letters.</p> <p>On IBM mainframe computers, the EBCDIC specification of characters may be used instead of ASCII.</p> |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## string type

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>stringType</i> is:</p> <p><b>string</b> [ ( <i>maxLength</i> ) ]</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Description | <p>Each variable whose type is a <i>stringType</i> can contain a sequence (a string) of characters. The length of this sequence must not exceed the <i>stringType</i>'s maximum length.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Example     | <pre> <b>var</b> name : <b>string</b> name := "Nancy" <b>var</b> licenceNumber : <b>string</b> ( 6 )         licenceNumber := "175AJN" </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Details     | <p>Strings can be assigned and they can be compared for both equality and for ordering. See also <i>string comparison</i> and <i>assignment statement</i>.</p> <p>Strings can be catenated (joined together) using the + operator and separated into substrings. See <i>catenation</i> and <i>substring</i>. String functions are provided to find the length of a string, to find where one string appears inside another, and to make repeated copies of a string all joined together. See <i>length</i>, <i>index</i>, and <i>repeat</i>.</p> <p>A string type written without a maximum length is limited to holding a maximum of 255 characters.</p> |



The *maxLength* of a string, if given as a part of the type, must be known at compile time, and must be at least 1 and at most 255. The maximum length of a string is given by *upper*, for example, *upper(licenceNumber)* is 6. See also *upper*.

In the declaration of a string that is a **var** formal parameter of a procedure or function, the *maxLength* can be written as an asterisk (\*). Here, the maximum length is taken to be that of the corresponding actual parameter, as in:

**procedure** *deblank* (**var** *s* : **string** (\*) ).

The star can also be used when the parameter is an array of strings.

See also *explicitStringConstants* for exact rules for writing string values such as "Nancy". See also **char**(*n*) and **char** types.

## strint      string-to-integer function

Syntax            **strint** ( *s* : **string** [ , *base* : **int** ] ) : **int**

Description      The **strint** function is used to convert a string to an integer. The integer is equivalent to string *s*. The number *base* parameter is optional, for example, **strint** ("-47") = -47. In Turing proper, the *base* is not allowed and is assumed to be 10.

String *s* must consist of a possibly null sequence of blanks, then an optional plus or minus sign, and finally a sequence of one or more digits. For number bases larger than 10, the digits can include a, b, c ... (alternately A, B, C ...) which represent the digit values 10, 11, 12 ... The *base*, if given, must be in the range 2 to 36 (36 because there are 10 base ten digits and 26 letters). For example, **strint** ("FF", 16) = 255.

The **intstr** function is the inverse of **strint**, so for any integer *i*,

**strint** ( **intstr** ( *i* ) ) = *i*.

See also            **chr**, **ord**, **intstr** and **strnat** functions.

## strintok string-to-integer function

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>strintok</b> ( <i>s</i> : <b>string</b> [ , <i>base</i> : <b>int</b> ] ) : <b>boolean</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Description | <p>The <b>strintok</b> function is used determine whether the <b>strint</b> function can be used to convert the string to an integer without causing an error. If the string can be successfully converted, then <b>strintok</b> returns <b>true</b>, otherwise it returns <b>false</b>.</p> <p>String <i>s</i> should consist of a possibly null sequence of blanks, then an optional plus or minus sign, and finally a sequence of one or more digits. For number bases larger than 10, the digits can include a, b, c ... (alternately A, B, C ...) which represent the digit values 10, 11, 12 ... If <i>s</i> is correctly constructed, then <b>strnatok</b> will return <b>true</b>, otherwise it returns <b>false</b>. The <i>base</i>, if given, must be in the range 2 to 36 (36 because there are 10 base ten digits and 26 letters). For example, <b>strintok</b> ("FF", 16) = <b>true</b>.</p> |
| See also    | <b>strint</b> function that does the actual conversion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## strnat string to natural number function

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>strnat</b> ( <i>s</i> : <b>string</b> [ , <i>base</i> : <b>int</b> ] ) : <b>nat</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Description | <p>The <b>strnat</b> function is used to convert a string to a natural number. The natural number is equivalent to string <i>s</i>. The number <i>base</i> parameter is optional, for example, <b>strnat</b>("47") = 47.</p> <p>String <i>s</i> must consist of a possibly null sequence of blanks, then an optional plus sign, and finally a sequence of one or more digits. For number bases larger than 10, the digits can include a, b, c ... (alternately A, B, C ...) which represent the digit values 10, 11, 12 ... The <i>base</i>, if given, must be in the range 2 to 36 (36 because there are 10 base ten digits and 26 letters). For example, <b>strnat</b>("FF", 16) = 255.</p> <p>The <b>natstr</b> function is the inverse of <b>strnat</b>, so for any natural number <i>n</i>, <b>strnat</b>( <b>natstr</b> (<i>n</i>)) = <i>n</i>.</p> <p>The <b>strnat</b> function is similar to <b>strint</b>, except that <b>strnat</b> handles values that are larger than <b>int</b> values and does not handle negative values.</p> |

See also the **chr**, **ord**, **intstr** and **strint** functions.

## strnatok string to natural number function

Syntax **strnatok** ( *s* : string [ , *base* : int ] ) : boolean

Description The **strnatok** function is used determine whether the **strnat** function can be used to convert the string to a natural number without causing an error. If the string can be successfully converted, then **strnatok** returns **true**, otherwise it returns **false**.

String *s* should consist of a possibly null sequence of blanks, then an optional plus sign, and finally a sequence of one or more digits. For number bases larger than 10, the digits can include a, b, c ... (alternately A, B, C ...) which represent the digit values 10, 11, 12 ... If *s* is correctly constructed, then **strnatok** will return **true**, otherwise it returns **false**. The *base*, if given, must be in the range 2 to 36 (36 because there are 10 base ten digits and 26 letters). For example, **strnatok** ("FF", 16) = **true**.

See also **strnat** function that does the actual conversion.

## strreal string-to-real function

Syntax **strreal** ( *s* : string ) : real

Description The **strreal** function is used to convert a string to a **real** number. For example, **strreal** ("2.5e1") will produce an approximation to the number 25.0.

String *s* must consist of a possibly null sequence of blanks, then an optional plus or minus sign and finally an explicit unsigned real or integer constant.

The **realstr**, **erealstr** and **frealstr** functions approximate the inverse of **strreal**, although round-off errors keep these from being exact inverses.

See also **realstr**, **erealstr**, **frealstr**, **intstr** and **strint** functions.

## strrealok string-to-real function

Syntax **strrealok** ( *s* : **string** ) : **boolean**

Description The **strrealok** function is used determine whether the **strreal** function can be used to convert the string to a real number without causing an error. If the string can be successfully converted, then **strrealok** returns **true**, otherwise it returns **false**.  
String *s* should consist of a possibly null sequence of blanks, then an optional plus or minus sign and finally an explicit unsigned real or integer constant. If it does so, then **strrealok** will return **true**, otherwise it returns **false**.

See also **strreal** function that does the actual conversion.

## subprogramHeader

Syntax A *subprogramHeader* is one of:

- (a) **procedure** [ **pervasive** ] *id*  
[ ([ *paramDeclaration* {, *paramDeclaration*} )]]
- (b) **function** [ **pervasive** ] *id*  
[ ([ *paramDeclaration* {, *paramDeclaration*} )]]  
[ *id* ] : *typeSpec*

Description A subprogram header is used to describe the interface to a subprogram. Subprogram headers are used within other language features such as subprogram types and external declarations.

Parameterless subprograms may use parentheses (with nothing between them), as is required in the C programming language. These parentheses can be used to disambiguate between the call to the subprogram (parentheses present) and a reference the subprogram (parentheses missing).

Suppose  $f$  is a parameterless subprogram declared without parentheses and  $g$  is a parameterless subprogram declared with parentheses. Their headers are:

```
procedure f
procedure g ()
```

In a program,  $f$  and  $g()$  are calls to these functions, while  $g$  is a reference to (not a call to) the procedure. There is no way to write a reference to  $f$ . When in doubt, use parentheses in the declaration, as in the case for  $g$ , so that calls always have parentheses and references always do not. A reference to a subprogram can be assigned to a subprogram variable. See subprogram type.

Example Specify that  $t$  is the type of procedure with a **var** integer parameter and a real parameter. See also *subprogramType*.

```
type t : procedure q (var j : int, y : real)
```

Details The keyword **pervasive** can be inserted just after **procedure** or **function**. When this is done, the subprogram is visible inside all subconstructs of the subprogram's scope. Without **pervasive**, the subprogram is not visible inside modules unless explicitly imported. Pervasive subprograms need not be imported. You can abbreviate **pervasive** as an asterisk (\*).

See also **pervasive**.

## subprogramType

Syntax A *subprogramType* is:

```
subprogramHeader
```

Description A variable or constant can contain a reference to a subprogram. The type of the variable or constant is a *subprogramType*. See also *subprogramHeader*.

Example In the following  $t$  is a subprogram type, and  $u$  is a variable of type  $t$  initialized to refer to procedure *rnd*.

```
procedure rnd (var i : int, x : real)
 i := round (x)
```

```

end rnd

type t : procedure q (var j : int, y : real)
var u : t := rnd % Procedure variable u refers to rnd
...
var j : int
u (j, 24.6) % Call procedure u referring to rnd
...
var v := u % Subprogram variable v initialized to u

```

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details | <p>The name of the subprogram, for example <i>q</i>, and the parameters, for example <i>i</i> and <i>x</i>, have no meaning in a subprogram type. They are present only because of the form of subprogram headers.</p> <p>If <i>v</i> is a variable or constant that refers to a subprogram, <i>v</i> can be called, compared for equality to other subprogram variables, assigned and passed as a parameter. Variable <i>v</i> is not an integer, string or pointer and cannot participate in their corresponding operations.</p> <p>A reference to a subprogram, rather than the code of the subprogram, is contained in a variable <i>v</i> whose type is a subprogram type. This implies that <b>addr</b> (<i>v</i>) is the address of the reference to subprogram, rather than the address of the subprogram. The address of the code is given by <i>#v</i>. See <b>cheat</b> for an explanation of the <i>#</i> operator.</p> <p>You cannot assign a reference to a subprogram exported from a class. This restriction exists because these subprograms are meaningless without an accompanying reference to an object.</p> <p>Many potential uses of subprogram variables are better programmed using classes and overriding exported subprograms. See <b>class</b>.</p> |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## subrangeType

|             |                                                                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>subrangeType</i> is:</p> <p style="text-align: center;"><i>expn .. expn</i></p>                                                                                                                                                                                                                |
| Description | <p>A subrange type defines a set of values, for example, the subrange 1 .. 4 consists of 1, 2, 3 and 4.</p>                                                                                                                                                                                            |
| Example     | <pre> var i : 1 .. 10          % i can be 1, 2 ... up to 10 type xRange : 0 .. 319   % Define integer subrange var pixels : array xRange of int                         % Array elements are                         % numbered 0, 1, ... 319 for k : xRange           % k ranges from 0 to 319 </pre> |

```
pixels (k) := 0
end for
```

**Details** A subrange must contain at least one element. In other words, the second expression (*expn*) must be at least as large as the first expression.

The lower bound of a subrange must be known at compile time. The upper bound is allowed to be a run time value only in one situation and that is when it gives the upper bound of an array being declared in a variable declaration, in other words when declaring a *dynamic* array.

Subranges are usually a subset of the integers, as in 1 .. 10. You can also have subranges of enumerated types and characters (the **char** type).

You can apply **lower** and **upper** to subrange types.

## substring of another string

**Syntax** A *substring* is one of:

- (a) *stringReference* ( *leftPosition* .. *rightPosition* )
- (b) *stringReference* ( *charPosition* )

**Description** A substring selects a part of another string. In form (a) the substring starts at the left position and runs to the right position. In form (b), the substring is only a single character. Turing support substrings of **char**(*n*) values.

**Example**

```
var word : string := "bring"
put word (2 .. 4) % Outputs rin
put word (3) % Outputs i
put word (2 .. *) % Outputs ring; the star (*) means
 % the end of the string.
put word (* - 2 .. * - 1) % Outputs in
```

**Details** The leftmost possible position in a string is numbered 1. The last position in a string can be written as an asterisk (\*). For example, *word* (2 .. \*) is equivalent to *word* (2 .. *length*(*word*)).

Each of *leftPosition*, *rightPosition*, and *charPosition* must have one of these forms:

- (a) *expn*
- (b) \*
- (c) \* - *expn*

The exact rules for the allowed values of *leftPosition* and *rightPosition* are:

- (1) *leftPosition* must be at least 1,
- (2) *rightPosition* must be at most *length (stringReference)*, and
- (3) the length of the selected substring must zero or more.

This specifically allows null substrings such as *word* (1, 0) in which *rightPosition* is 0 and *word* (6, 5) in which *leftPosition* is one more than **length** (*stringReference*).

Note that substrings are not assignable. For example, if *s* is a string, the statement *s* (3) := "a" is illegal in Turing.

Turing supports substrings of **char**(*n*) values. See **char**(*n*). If a substring of **char**(*n*) value *t* has two operands, as in *t*(2..77), the result type of this operation is a **string**. If the substring has one operand, as in *t*(7), this becomes, in effect, a subscript into an array of characters. The result is a reference to a **char**, which can be assigned to or passed to a **var** parameter.

See also **string**, **char**, **char**(*n*), *explicitStringConstant*, *explicitCharConstant*, catenation and **length**.

## SUCC successor function

Syntax **succ** ( *expn* )

Description The **succ** function accepts an integer, character or an enumerated value and returns the integer plus one, the next character, or the next value in the enumeration. For example, **succ** (7) is 8.

Example This part of a Turing program fills up array *a* with the enumerated values *green*, *yellow*, *red*, *green*, *yellow*, *red*, etc.

```

type colors : enum (green, yellow, red)
var a : array 1 .. 100 of colors
var c : colors := colors . green
for i : 1 .. 100
 a (i) := c
 if c = colors . red then
 c := colors . green
 else
 c := succ (c)
 end if
end for

```

Details You cannot apply **succ** to the last value of an enumeration.

See also the **pred**, **lower** and **upper** functions.



# Sys

|              |                                                                                                                                                                                                                                                                                                                                |                                                               |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that deal with the operating system directly (getting the process id, getting run time arguments and executing commands in the operating system, etc.).</p> <p>All routines in the <b>Sys</b> unit are exported qualified (and thus must be prefaced with "<b>Sys.</b>").</p> |                                                               |
| Entry Points | <b>GetEnv</b>                                                                                                                                                                                                                                                                                                                  | Gets a string associated with an environment variable.        |
|              | <b>GetPid</b>                                                                                                                                                                                                                                                                                                                  | Gets the current process ID for Turing.                       |
|              | <b>Exec</b>                                                                                                                                                                                                                                                                                                                    | Executes a program using the operating system.                |
|              | <b>Nargs</b>                                                                                                                                                                                                                                                                                                                   | Gets the number of run time arguments (exported unqualified). |
|              | <b>FetchArg</b>                                                                                                                                                                                                                                                                                                                | Gets a specified run time argument (exported unqualified).    |

## Sys.Exec



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sys.Exec</b> ( <i>command</i> : <b>string</b> ) : <b>int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Description | <p>The <b>Sys.Exec</b> function is used to execute the shell (operating system) <i>command</i>, as if it were typed at the terminal. The function returns the exit code from the <i>command</i>. In general, a return code of 0 (zero) means no detected errors. However, the <i>command</i> could return any exit code. If <b>Sys.Exec</b> returns the predefined constant <i>sysExecFailed</i>, then <i>command</i> was not executed and the program should use <b>Last.Error</b> or <b>Last.ErrorStr</b> to determine the reason for failure.</p> |
| Example     | <p>This program creates a directory listing when run under DOS on an IBM PC compatible computer. The same program will run under UNIX by changing "<i>dir</i>" to "<i>ls</i>".</p> <pre> var retCode : int retCode := Sys.Exec ( "dir" ) if retCode = sysExecFailed then </pre>                                                                                                                                                                                                                                                                      |

```

 put "The Sys.Exec call failed"
 put "Error: ", Error.LastMsg
else
 put "The call returned with a code of: ", retCode
end if

```

|          |                                                                                                                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | When the <b>Sys.Exec</b> procedure is used, the executing program remains in memory while the command is executing, and once execution of the command is finished, control returns to the original program. |
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Sys.Exec</b> , not by calling <b>Exec</b> .                                                                             |
| See also | <b>Sys.Nargs</b> , <b>Sys.FetchArg</b> and <b>Sys.GetEnv</b> functions.                                                                                                                                     |

## Sys.FetchArg All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>System.FetchArg ( i : int ) : string</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Description | <p>The <b>Sys.FetchArg</b> function is used to access the <i>i</i>-th argument that has been passed to a program from the command line. For example, if the program is run from the Turing environment using</p> <pre style="margin-left: 40px;">:r file1 file2</pre> <p>then <b>Sys.FetchArg</b> (2) will return "file2". If a program called <i>prog.x</i> is run under UNIX using this command:</p> <pre style="margin-left: 40px;">prog.x file1 file2</pre> <p>the value of <b>Sys.FetchArg</b>(2) will similarly be "file2".</p> <p>The <b>Sys.Nargs</b> function, which gives the number of arguments passed to the program, is usually used together with the <b>Sys.FetchArg</b> function. Parameter <i>i</i> passed to <b>Sys.FetchArg</b> must be in the range 0 .. <b>Sys.Nargs</b>. The 0-th argument is the name of the running program.</p> |
| Example     | <p>This program lists its own name and its arguments.</p> <pre> put "The name of this program is : ", Sys.FetchArg ( 0 ) for i : 1 .. Sys.Nargs     put "Argument ", i, " is ", Sys.FetchArg ( i ) end for </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

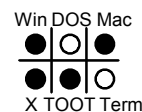
|          |                                                                                                                                         |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Sys.FetchArg</b> , not by calling <b>FetchArg</b> . |
| See also | <b>Sys.Nargs</b>                                                                                                                        |

## Sys.GetEnv



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sys.GetEnv</b> ( <i>symbol</i> : <b>string</b> ) : <b>string</b>                                                                                                                                                                                                                                                                                                                                                                                                          |
| Description | The <b>Sys.GetEnv</b> function is used to access the environment string whose name is <i>symbol</i> . These strings are determined by the shell (command processor) or the program that caused your program to run. See also the <b>Sys.Nargs</b> and <b>Sys.FetchArg</b> functions.                                                                                                                                                                                         |
| Example     | On a PC, this retrieves the environment variable USERLEVEL and prints extra instructions if USERLEVEL had been set to NOVICE. USERLEVEL can be set to NOVICE with the command <b>SET USERLEVEL = NOVICE</b> in the <b>autoexec.bat</b> file or in any batch file.<br><br><pre> <b>const</b> userLevel : <b>string</b> userLevel := <b>Sys.GetEnv</b> ("USERLEVEL") <b>if</b> userLevel = "NOVICE" <b>then</b> ...           % put a set of instructions <b>end if</b> </pre> |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Sys.GetEnv</b> , not by calling <b>GetEnv</b> .                                                                                                                                                                                                                                                                                                                                          |

## Sys.GetPid




|        |                                |
|--------|--------------------------------|
| Syntax | <b>Sys.GetPid</b> : <b>int</b> |
|--------|--------------------------------|

|             |                                                                                                                                                                                                                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | <p>The <b>Sys.GetPid</b> function is used to determine the I.D. (number) that identifies the current operating system task (process). Beware that there are processes, activated by the <b>fork</b> statement, that are independent of the operating systems tasks.</p> <p>Under UNIX, the number is used, for example, for creating a unique name of a file.</p> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Sys.GetPid</b>, not by calling <b>GetPid</b>.</p>                                                                                                                                                                                                                      |
| See also    | <b>Sys.Nargs</b> , <b>Sys.FetchArg</b> and <b>Sys.GetEnv</b> .                                                                                                                                                                                                                                                                                                    |

## Sys.Nargs All


|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Sys.Nargs : int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Description | <p>The <b>Sys.Nargs</b> function is used to determine the number of arguments that have been passed to a program from the command line. For example, if the program is run from the Turing environment using</p> <pre style="margin-left: 40px;">:r file1 file2</pre> <p>then <b>Sys.Nargs</b> will return 2. If a program called <i>prog.x</i> is run under UNIX using this command:</p> <pre style="margin-left: 40px;">prog.x file1 file2</pre> <p>the value of <b>Sys.Nargs</b> will similarly be 2.</p> <p>The <b>Sys.Nargs</b> function is usually used together with the <b>Sys.FetchArg</b> function to access the arguments that have been passed to the program.</p> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Sys.Nargs</b>, not by calling <b>Nargs</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| See also    | <b>Sys.FetchArg</b> for an example of the use of <b>Sys.Nargs</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## sysclock      millisecs used procedure

Win DOS Mac  
  
 X TOOT Term

- Syntax**                **sysclock ( var *c* : int )**
- Description**            The **sysclock** statement is used on a multitasking system such as UNIX to determine the amount of time that has been used by this program (process). Variable *c* is assigned the number of central processor milliseconds assigned to this program. This is of little use on a personal computer, where **sysclock** returns the same value as **clock**.
- Example**                On a UNIX system, this program tells you how much time it has used.
- ```
var timeUsed : int
sysclock ( timeUsed )
put "This program has used ", timeUsed,
    " milliseconds of CPU time"
```
- 61 **delay**, **time**, **clock**, **wallclock** and **date** statements.
 See also predefined unit **Time**.

system statement

Win DOS Mac

 X TOOT Term

- Syntax** **system (*command* : string, var *ret* : int)**
- Description** The **system** statement is used to execute the shell (operating system) *command*, as if it were typed at the terminal. The return code is in *ret*. A return code of 0 (zero) means no detected errors. A return code of 127 means the command processor could not be accessed. A return code of 126 means the command processor did not have room to run on the PC.
- Example** This program creates a directory listing when run under DOS on an IBM PC compatible computer. The same program will run under UNIX by changing "*dir*" to "*ls*".
- ```
var success : int
system ("dir", success)
if success not= 0 then
 if success = 127 then
```

```

 put "Sorry, can't find 'command.com'"
 elsif success = 126 then
 put "Sorry, no room to run 'dir'"
 else
 put "Sorry, 'dir' did not work"
 end if
end if
end if

```

Details            When the **system** procedure is used, the executing program usually remains in memory while the system command is executing, and once execution of the system command is finished, control returns to the original program. However, on the PC, there is variant of the **system** procedure that allows "chaining". This means that when the system command is executed, the originally running program is "thrown away" (i.e. removed from memory). When the executed program terminates, one is returned to DOS.

To chain another program, one prepends "chain:" to the start *command*.

i.e.    **system** ("chain:myprog.exe", *retCode*)

Note that this command is "hazardous". Specifically, if you call it from Turing (as opposed to a program compiled with TComp) and you have not saved your source file, **you will lose it!** Turing will be removed from memory without any warning when the **system** procedure is executed. Likewise any open files will be closed instantly. This means there is a danger if all files were not properly closed before the **system** procedure was called.

The "chain:" command is often used for starting menu programs, where the user selects a program to run and doesn't want Turing to remain in memory. It can also be used with extraordinarily large Turing programs that can be split into different parts. By using TComp and compiling each part separately, one can have each program call the other and never have all parts in memory at once.

Example            This program uses chaining to launch one of several possible programs based on user choice. It gives an error if for some reason the **system** command fails to work. It assumes that c:\chemistry.exe, c:\math.exe, c:\english.exe and c:\history.exe already exist.

```

var choice, success : int
put "Enter the subject (1-4): "..
get choice
var command : string
case choice of
 % Note the use of the double backslash in the file name. This
 % is because the backslash is a special character in Turing (as
 % in \t for tab and \n for a newline). To get a single backslash,
 % one uses \\.
 label 1 : command := "c:\\chemistry.exe"
 label 2 : command := "c:\\math.exe"
 label 3 : command := "c:\\english.exe"
 label 4 : command := "c:\\history.exe"
 label : put "Choice must be from 1-4."
 assert false % Wasn't a 1-4. Terminate.
end case

```

```

end case
system ("chain:" + command, success)
% If I reach this line, the system command failed and one should give
% an error message.
put "System called failed."
put "Program \"", command, "\" couldn't be run."
assert false % Terminate the program

```

|          |                                                                                                      |
|----------|------------------------------------------------------------------------------------------------------|
| Details  | Here are the possible errors under PC-Turing                                                         |
|          | -1 Not enough memory to load COMMAND.COM                                                             |
|          | -2 Not enough memory to run command                                                                  |
|          | -3 Argument list greater than 128 bytes or environment info is greater than 32k                      |
|          | -4 Couldn't find COMMAND.COM                                                                         |
|          | -5 COMMAND.COM corrupt                                                                               |
|          | -6 -noshell option is selected, the system procedure is disallowed                                   |
| See also | <b>nargs</b> , <b>fetcharg</b> and <b>getenv</b> functions.<br>See also predefined unit <b>Sys</b> . |

## tag statement

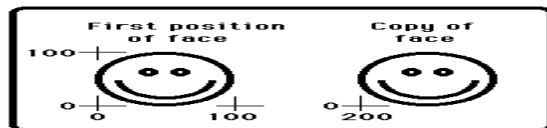
|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | A <i>tagStatement</i> is:<br><b>tag</b> <i>union VariableReference</i> , <i>expn</i>                                                                                                                                                                                                                                                                                                                                                                                  |
| Description | A <b>tag</b> statement is a special-purpose assignment that is used for changing the tag of a <b>union</b> variable.                                                                                                                                                                                                                                                                                                                                                  |
| Example     | In this example, the tag field of <b>union</b> variable <i>v</i> is set to be <i>passenger</i> , thereby activating the <i>passenger</i> field of <i>v</i> .<br><pre> type vehicleInfo :   union kind : passenger .. recreational     label passenger :       cylinders : 1..16     label farm :       farmClass : string ( 10 )     label : % No fields for "otherwise" clause   end union var v : vehicleInfo ... tag v, passenger % Activate passenger part </pre> |

|          |                                                                                                                                                                                                                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | <p>A <b>tag</b> statement is the only way to modify the tag field of a <b>union</b> variable (other than by assigning an entire union value to the union variable).</p> <p>You cannot access a particular set of fields of a <b>union</b> unless the tag is set to match the corresponding label value.</p> |
| See also | <b>union</b> types.                                                                                                                                                                                                                                                                                         |

## takepic graphics procedure

Pixel graphics only

|             |                                                                                                                                                                                                                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>takepic</b> ( <i>x1</i> , <i>y1</i> , <i>x2</i> , <i>y2</i> : <b>int</b> , <b>var</b> <i>buffer</i> : <b>array</b> 1 .. * <b>of int</b> )                                                                                                                                                                                                                      |
| Description | The <b>takepic</b> procedure is used to record the pixel values in a rectangle, with left bottom and right corners of ( <i>x1</i> , <i>y1</i> ) and ( <i>x2</i> , <i>y2</i> ), in the buffer array. This requires a sufficiently-large buffer (see <b>sizepic</b> ). The <b>drawpic</b> procedure is used to make copies of the recorded rectangle on the screen. |
| Example     | After drawing a happy face, this program copies the face to a new location.                                                                                                                                                                                                                                                                                       |



```

setscreen ("graphics")
... draw happy face in the box (0,0) to (100,100) ...
% Create buffer big enough to hold happy face
var face : array 1 .. sizepic (0, 0, 100, 100) of int
% Copy picture into the buffer, which is the face array
takepic (0, 0, 100, 100, face)
% Redraw the picture with its left bottom at (200,0)
drawpic (200, 0, face, 0)

```

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | <p>The integer values that <b>takepic</b> places in the buffer can be read or written (using the <b>read</b> and <b>write</b> statements). Unfortunately, if a value happens to be the pattern used to represent the uninitialized value (the largest negative number the hardware can represent) assignment (by:=) and <b>put</b> of the individual integer values in the buffer will fail.</p> <p>The screen should be in a "graphics" mode. See the <b>setscreen</b> procedure for details. If the screen is not in a "graphics" mode, it will automatically be set to "graphics" mode.</p> |
| See also | <b>sizepic</b> and <b>drawpic</b> . See also <b>setscreen</b> , <b>maxx</b> , <b>maxy</b> , <b>drawdot</b> , <b>drawline</b> , <b>drawbox</b> , and <b>drawoval</b> .                                                                                                                                                                                                                                                                                                                                                                                                                          |



See also predefined unit **Pic**.

## tell file statement

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | An <i>tellStatement</i> is:<br><b>tell</b> : <i>fileNumber</i> , <i>filePositionVar</i>                                                                                                                                                                                                                                                                                                                                                              |
| Description | The <b>tell</b> statement sets <i>filePositionVar</i> , whose type must be <b>int</b> , to the current offset in bytes from the beginning of the specified file. The <i>fileNumber</i> must specify a file that is open with <b>seek</b> capability (or else a program argument file that is implicitly opened). The <b>tell</b> statement is useful for recording the file position of a certain piece of data for later access using <b>seek</b> . |
| Example     | This example shows how to use <b>tell</b> to record the location of a record in a file. This location is later used by <b>seek</b> to allow the record to be read.                                                                                                                                                                                                                                                                                   |

```
var employeeRecord :
 record
 name : string (30)
 pay : int
 dept : 0 .. 9
 end record
var fileNo : int
var location : int
open : fileNo, "payroll", write, seek
...
tell : fileNo, location % Make note of this location
write : fileNo, employeeRecord % Write record at this location
...
seek : fileNo, location % Go back to location
read : fileNo, employeeRecord % Read the record
 % that was previously written
```

See also        the **read**, **write**, **open**, **close**, **seek**, **get** and **put** statements.

# Text

|              |                                                                                                                                                                                                                                                                                                                                                 |                                                                                                 |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that handle character (text) output on the screen (i.e. output using <b>put</b>).</p> <p>All routines in the <b>Text</b> unit are exported qualified (and thus must be prefaced with "<b>Text.</b>") with the exception of <b>maxrow</b> and <b>maxcol</b> which are exported unqualified.</p> |                                                                                                 |
| Entry Points | <b>Cls</b>                                                                                                                                                                                                                                                                                                                                      | Clears the screen to the text background color.                                                 |
|              | <b>Color</b>                                                                                                                                                                                                                                                                                                                                    | Sets the text color used by <b>put</b> .                                                        |
|              | <b>Colour</b>                                                                                                                                                                                                                                                                                                                                   | Sets the text color used by <b>put</b> .                                                        |
|              | <b>ColorBack</b>                                                                                                                                                                                                                                                                                                                                | Sets the text background color used by <b>put</b> .                                             |
|              | <b>ColourBack</b>                                                                                                                                                                                                                                                                                                                               | Sets the text background color used by <b>put</b> .                                             |
|              | <b>Locate</b>                                                                                                                                                                                                                                                                                                                                   | Moves the cursor to the specified row and column.                                               |
|              | <b>LocateXY</b>                                                                                                                                                                                                                                                                                                                                 | Moves the cursor to the cursor location closest to a specified pixel position.                  |
|              | <b>maxcol</b>                                                                                                                                                                                                                                                                                                                                   | The number of columns on the screen (exported unqualified).                                     |
|              | <b>maxrow</b>                                                                                                                                                                                                                                                                                                                                   | The number of rows on the screen (exported unqualified).                                        |
|              | <b>WhatRow</b>                                                                                                                                                                                                                                                                                                                                  | Returns the current cursor row.                                                                 |
|              | <b>WhatCol</b>                                                                                                                                                                                                                                                                                                                                  | Returns the current cursor column.                                                              |
|              | <b>WhatColor</b>                                                                                                                                                                                                                                                                                                                                | Returns the current text color.                                                                 |
|              | <b>WhatColour</b>                                                                                                                                                                                                                                                                                                                               | Returns the current text color.                                                                 |
|              | <b>WhatColorBack</b>                                                                                                                                                                                                                                                                                                                            | Returns the current text background color.                                                      |
|              | <b>WhatColourBack</b>                                                                                                                                                                                                                                                                                                                           | Returns the current text background color.                                                      |
|              | <b>WhatChar</b>                                                                                                                                                                                                                                                                                                                                 | Returns the character, color and background color of a character at a specified row and column. |

# Text.Cls



|             |                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.Cls</b>                                                                                                                                                                                        |
| Description | The <b>Text.Cls</b> (clear screen) procedure is used to blank the screen to the text background color. The cursor is set to the top left (to row 1, column 1).                                         |
| Details     | The screen should be in a " <i>screen</i> " or " <i>graphics</i> " mode. If the screen mode has not been set, it will automatically be set to " <i>screen</i> " mode. See <b>View.Set</b> for details. |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Text.Cls</b> , not by calling <b>Cls</b> .                                                                         |

# Text.Color



|             |                                                                                                                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.Color</b> ( <i>Color</i> : int )                                                                                                                                                                                                                                                                           |
| Description | The <b>Text.Color</b> procedure is used to change the currently-active color. This is the color of characters that are to be <b>put</b> on the screen. The alternate spelling is <b>Text.Colour</b> .                                                                                                              |
| Example     | <p>This program prints out the message "Bravo" three times, each in a different color.</p> <pre>View.Set ( "graphics" ) for i : 1 .. 3   Text.Color ( i )   put "Bravo" end for</pre>                                                                                                                              |
| Example     | <p>This program prints out a message. The color of each letter is different from the preceding letter. For letter number <i>i</i> the color number is <i>i</i> mod maxcolor + 1. This cycles repeatedly through all the available colors.</p> <pre>View.Set ( "screen" ) const message := "Happy New Year!!"</pre> |

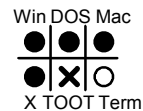
```

for i : 1 .. length (message)
 Text.Color (i mod maxcolor + 1)
 put message (i) ..
end for

```

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | <p>In "screen" mode on the IBM PC, the color specified can actually range from 0 - 31. The upper 16 colors (16-31) are the same as the lower 16, except that they blink.</p> <p>See <b>View.Set</b> for the number of colors available in the various "graphics" modes.</p> <p>The screen should be in a "screen" or "graphics" mode. If the screen mode has not been set, it will automatically be set to "screen" mode. See <b>View.Set</b> for details.</p> |
| Status   | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Text.Color</b>, not by calling <b>Color</b>.</p>                                                                                                                                                                                                                                                                                                                    |
| See also | <b>Text.ColorBack</b> , <b>Text.WhatColor</b> , <b>Text.WhatChar</b> and <b>View.maxcolor</b> .                                                                                                                                                                                                                                                                                                                                                                |

## Text.ColorBack



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.ColorBack</b> ( <i>Color</i> : int )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | <p>The <b>Text.ColorBack</b> procedure is used to change the current text background color. The alternate spelling is <b>Text.ColourBack</b>.</p> <p>The <b>Text.ColorBack</b> procedure sets the text background color to the specified color. This is the color that surrounds characters when they are <b>put</b> onto the screen. On an IBM PC in "screen" mode, the color can be from 0 - 7. (You can not have the upper 8 colors as text background colors. On UNIX dumb terminals, <b>Text.ColorBack</b>(1) turns on highlighting and <b>Text.ColorBack</b>(0) turns it off. On other systems, this procedure may have no effect.</p> |
| Example     | <p>Since this program is in "screen" mode, changing the background color has no immediately observable effect. When the message "Greetings" is output, the background surrounding each letter will be red.</p> <pre> View.Set ( "screen" ) ... Text.ColorBack ( red )   put "Greetings" </pre>                                                                                                                                                                                                                                                                                                                                               |

|          |                                                                                                                                                                            |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | The screen should be in a "screen" or "graphics" mode. If the screen mode has not been set, it will automatically be set to "screen" mode. See <b>View.Set</b> for details |
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Text.ColorBack</b> , not by calling <b>ColorBack</b> .                                 |
| See also | <b>Text.Color</b> and <b>Text.WhatColorBack</b> .                                                                                                                          |

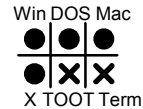
## Text.Locate



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.Locate</b> ( <i>row, column : int</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Description | The <b>Text.Locate</b> procedure is used to move the cursor so that the next output from <b>put</b> will be at the given row and column. Row 1 is the top of the screen and column 1 is the left side of the screen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Example     | <p>This program outputs stars of random colors to random locations on the screen. The variable <i>colr</i> is purposely spelled differently from the word <i>color</i> to avoid the procedure of that name (used to set the color of output). The row number is purposely chosen so that it is one less than <b>maxrow</b>. This avoids the scrolling of the screen which occurs when a character is placed in the last column of the last row.</p> <pre> <b>View.Set</b> ("screen") <b>var</b> row, column, colr : <b>int</b> <b>loop</b>   row := <b>Rand.Int</b> (1, <b>maxrow</b>)   column := <b>Rand.Int</b> (1, <b>maxcol</b>)   colr := <b>Rand.Int</b> (0, <b>maxcolor</b>)   <b>Text.Color</b> (colr)   <b>Text.Locate</b> (row, column )   <b>put</b> "*" .. % Use dot-dot to avoid clearing end of line <b>end loop</b> </pre> |
| Details     | <p>The <b>Text.Locate</b> procedure is used to locate the next output based on row and column positions. See also the <b>Text.LocateXY</b> procedure which is used to locate the output based x and y positions, where x=0, y=0 is the left bottom of the screen.</p> <p>The screen should be in a "screen" or "graphics" mode. See the <b>View.Set</b> procedure for details. If the screen is not in one of these modes, it will automatically be set to "screen" mode.</p>                                                                                                                                                                                                                                                                                                                                                              |

|          |                                                                                                                                      |
|----------|--------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Text.Locate</b> , not by calling <b>Locate</b> . |
| See also | <b>View.Set</b> and <b>Draw.Dot</b> .                                                                                                |

## Text.LocateXY



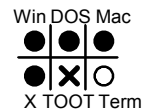
|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.LocateXY</b> ( <i>x</i> , <i>y</i> : int )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Description | The <b>Text.LocateXY</b> procedure is used to move the cursor so that the next output from <b>put</b> will be at approximately ( <i>x</i> , <i>y</i> ). The exact location may be somewhat to the left of <i>x</i> and below <i>y</i> to force alignment to a character boundary.                                                                                                                                                                                                                                                                                                            |
| Example     | This program outputs <i>Hello</i> starting at approximately (100, 50) on the screen.<br><br><pre> <b>View.Set</b> ("graphics") <b>Text.LocateXY</b> ( 100, 50 ) <b>put</b> "Hello" </pre>                                                                                                                                                                                                                                                                                                                                                                                                    |
| Details     | <p>The <b>Text.LocateXY</b> procedure is used to locate the next output based on <i>x</i> and <i>y</i> positions, where the position <i>x</i>=0, <i>y</i>=0 is the left bottom of the screen. See also the <b>Text.Locate</b> procedure which is used to locate the output-based row and column positions, where row 1 is the top row and column 1 is the left column.</p> <p>The screen should be in a "<i>graphics</i>" mode. See the <b>View.Set</b> procedure for details. If the screen is not in a "<i>graphics</i>" mode, it will automatically be set to "<i>graphics</i>" mode.</p> |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Text.LocateXY</b> , not by calling <b>LocateXY</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| See also    | <b>View.Set</b> and <b>Draw.Dot</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

# Text.maxcol



|             |                                                                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>maxcol : int</b>                                                                                                                                                                                                 |
| Description | The <b>maxcol</b> function is used to determine the number of columns on the screen.                                                                                                                                |
| Example     | This program outputs the maximum column number.<br><b>put</b> "Number of columns on the screen is ", <b>maxrow</b>                                                                                                  |
| Details     | For IBM PC compatibles as well as most UNIX dumb terminals, in " <i>text</i> " or " <i>screen</i> " mode, <b>maxcol</b> = 80. For the default IBM PC compatible " <i>graphics</i> " mode (CGA), <b>maxcol</b> = 40. |
| Status      | Exported unqualified.<br>This means that you can call the function by calling <b>maxcol</b> or by calling <b>Text.maxcol</b> .                                                                                      |
| See also    | <b>Text.Locate</b> procedure for an example of the use of <b>maxcol</b> .                                                                                                                                           |

# Text.maxrow



|             |                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>maxrow : int</b>                                                                                                            |
| Description | The <b>maxrow</b> function is used to determine the number of rows on the screen.                                              |
| Example     | This program outputs the maximum row number.<br><b>put</b> "Number of rows on the screen is ", <b>maxrow</b>                   |
| Details     | For IBM PC compatibles, <b>maxrow</b> = 25. For many UNIX dumb terminals, <b>maxrow</b> = 24.                                  |
| Status      | Exported unqualified.<br>This means that you can call the function by calling <b>maxrow</b> or by calling <b>Text.maxrow</b> . |

See also **Text.Locate** procedure for an example of the use of **maxrow**.

## Text.WhatChar

Win DOS Mac  
X O X  
X X O  
X TOOT Term

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.WhatChar</b> ( <i>row</i> , <i>col</i> : <b>int</b> , <b>var</b> <i>ch</i> : <b>char</b> ,<br><b>var</b> <i>foreColor</i> , <i>backColor</i> : <b>int</b> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Description | The <b>whattextchar</b> function is used to determine the character and text colours on the screen at the specified location. This function can only be used in DOS and in " <i>screen</i> " mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Example     | <p>This program outputs a message and then changes the foreground color (the color of the letters) of the message to color number 1 and the background color (surrounding each letter) to color number 7. The actual message (each letter) is not changed.</p> <pre><b>View.Set</b> ( "screen" )<br/><b>const</b> message := "Happy New Year!!"<br/><b>put</b> message<br/>...<br/><b>var</b> ch : <b>char</b><br/><b>var</b> fore, back : <b>int</b><br/><br/><b>for</b> column : 1 .. <b>length</b> ( message )<br/>  <b>locate</b> ( 1, column )<br/>  <b>Text.WhatChar</b> (1, column, ch, fore, back)<br/>  <b>Text.Color</b> ( 1 )      % Color of letter<br/>  <b>Text.ColorBack</b> ( 7 )  % Color around letter<br/>  <b>put</b> ch ..             % Use same letter<br/><b>end for</b></pre> |
| Details     | The <b>Text.WhatChar</b> procedure is meaningful only in " <i>screen</i> " mode. In " <i>graphics</i> " mode, the concept of <i>text</i> on the screen is replaced by the concept of <i>pixels</i> on the screen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Status      | Exported qualified.<br><br>This means that you can only call the function by calling <b>Text.WhatChar</b> , not by calling <b>WhatChar</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| See also    | <b>View.Set</b> which describes modes. See also <b>Text.Color</b> and <b>Text.ColorBack</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

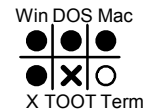


# Text.WhatCol



|             |                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.WhatCol : int</b>                                                                                                                                                                                                                              |
| Description | The <b>Text.WhatCol</b> function is used to determine the cursor position's column.                                                                                                                                                                    |
| Example     | This program outputs <i>The current row is 5, the current column is 15.</i><br><b>Text.Locate</b> ( 5, 10 )<br><b>put</b> "12345"..<br><b>put</b> "The current row is", <b>Text.WhatRow</b><br><b>put</b> "The current column is", <b>Text.WhatCol</b> |
| Details     | The screen should be in a "screen" or "graphics" mode. <b>Text.WhatCol</b> functions properly even if the cursor is invisible.                                                                                                                         |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Text.WhatCol</b> , not by calling <b>WhatCol</b> .                                                                                                                 |
| See also    | the <b>Text.WhatRow</b> function, which is used to determine the cursor row.<br>See also the <b>Text.Locate</b> , <b>Text.maxrow</b> and <b>Text.maxcol</b> procedure.                                                                                 |

# Text.WhatColor



|             |                                                                                                                                                                                                                                 |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.WhatColor : int</b>                                                                                                                                                                                                     |
| Description | The <b>Text.WhatColor</b> function is used to determine the current text (foreground) color, ie., the color used for characters that are output using <b>put</b> . The alternate spelling is <b>Text.WhatColour</b> .           |
| Example     | This program outputs the currently-active color number. The message is also given in the currently-active color.<br><b>View.Set</b> ("graphics")<br>...<br><b>put</b> "This writing is in color number ", <b>Text.WhatColor</b> |

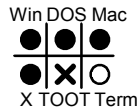
|          |                                                                                                                                            |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | The screen should be in a "screen" or "graphics" mode. See <b>View.Set</b> for details.                                                    |
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Text.WhatColor</b> , not by calling <b>WhatColor</b> . |
| See also | the <b>Text.Color</b> procedure, which is used to set the color. See also <b>Text.ColorBack</b> and <b>Text.WhatColorBack</b> .            |

## Text.WhatColorBack



|             |                                                                                                                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.WhatColorBack : int</b>                                                                                                                                                                                                                                                                            |
| Description | The <b>Text.WhatColorBack</b> function is used to determine the current text background color. The alternate spelling is <b>whatcolourback</b> .                                                                                                                                                           |
| Example     | This program outputs the currently-active background color number. The background color of the message is determined by this number.<br><br><pre> <b>View.Set</b> ("screen") ... <b>put</b> "The background of this writing"            <b>put</b> "is in color number ", <b>Text.WhatColorBack</b> </pre> |
| Details     | The screen should be in a "screen" or "graphics" mode. Beware that the meaning of background color is different in these two modes. See <b>Text.ColorBack</b> for details.                                                                                                                                 |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Text.WhatColorBack</b> , not by calling <b>WhatColorBack</b> .                                                                                                                                                         |
| See also    | <b>Text.Color</b> and <b>Text.WhatColor</b> .                                                                                                                                                                                                                                                              |

# Text.WhatRow



|             |                                                                                                                                                                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Text.WhatRow : int</b>                                                                                                                                                                                                                                      |
| Description | The <b>Text.WhatRow</b> function is used to determine the cursor position's row.                                                                                                                                                                               |
| Example     | <p>This program outputs <i>The current row is 5, the current column is 15.</i></p> <pre> <b>Text.Locate</b> ( 5, 10 ) <b>put</b> "12345".. <b>put</b> "The current row is", <b>Text.WhatRow</b> <b>put</b> "The current column is", <b>Text.WhatCol</b> </pre> |
| Details     | The screen should be in a "screen" or "graphics" mode. <b>Text.WhatRow</b> functions properly even if the cursor is invisible.                                                                                                                                 |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Text.WhatRow</b>, not by calling <b>WhatRow</b>.</p>                                                                                                                |
| See also    | the <b>Text.WhatCol</b> function, which is used to determine the cursor column. See also the <b>Text.Locate</b> , <b>Text.maxrow</b> and <b>Text.maxcol</b> procedure.                                                                                         |

# Time



|              |                                                                                                                                                                                                                                                |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that handle anything to do with time, either as a date or as a timer.</p> <p>All routines in the <b>Time</b> unit are exported qualified (and thus must be prefaced with "<b>Time.</b>").</p> |
| Entry Points | <p><b>Sec</b> Returns the number of seconds since 1/1/1970 00:00:00 GMT.</p> <p><b>Date</b> Returns the current date and time as a string.</p> <p><b>SecDate</b> Converts a number of seconds into a date / time string.</p>                   |

|                   |                                                                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DateSec</b>    | Converts a date / time string to a number of seconds.                                                                                       |
| <b>SecParts</b>   | Converts the number of seconds since 1/1/1970 00:00:00 GMT into a day of month, month, year, day of week, hour, minute and second integers. |
| <b>PartsSec</b>   | Converts a day of month, month, year, hour, minute and second integers into the number of seconds since 1/1/1970 00:00:00 GMT.              |
| <b>Elapsed</b>    | Returns the number of milliseconds elapsed since the program started to run.                                                                |
| <b>ElapsedCPU</b> | Returns the number of milliseconds of CPU time elapsed since the program started to run.                                                    |
| <b>Delay</b>      | Sleeps for a specified number of milliseconds.                                                                                              |

## Time.Date All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.Date : string</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Description | The <b>Time.Date</b> function returns the current date and time as a string. The returned string in the format " <i>dd mmm yy hh:mm:ss</i> ", where <i>mmm</i> is the first 3 characters of the month, e.g., " <i>Apr</i> ". For example, if the date is Christmas 1989 at 9:02:37 in the morning, <b>Time.Date</b> will return " <i>25 Dec 89 09:02:37</i> ". Twenty-four hour time is used, so eleven thirty at night the same day would return " <i>25 Dec 89 23:30:00</i> ".                                                            |
| Example     | <p>This program greets you and tells you the date and time.</p> <pre> var theDateTime, theDate, theTime : string theDateTime := Time.Date theDate := theDateTime (1 .. 9) theTime := theDateTime (11 .. *) put "Greetings!! The date and time today is ", Time.Date </pre>                                                                                                                                                                                                                                                                  |
| Details     | <p>Be warned that on some computers, such as IBM PC compatibles or Apple Macintoshes, the date may not be set correctly in the operating system; in that case, the <b>Time.Date</b> procedure will give incorrect results.</p> <p>The string form of the date can be converted to a numeric form for comparison purposes using the <b>Time.DateSec</b> function. The numeric form can be converted to a string using the <b>Time.SecDate</b> function. The numeric form of the time can be obtained using the <b>Time.Sec</b> function.</p> |

|          |                                                                                                                                  |
|----------|----------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Time.Date</b> , not by calling <b>Date</b> . |
| See also | <b>Time.Sec</b> , <b>Time.DateSec</b> and <b>Time.SecDate</b> functions.                                                         |

## Time.DateSec All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.DateSec</b> ( <i>dateString</i> : <b>string</b> ) : <b>int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Description | <p>The <b>Time.DateSec</b> function is used to convert a date and time string into a number, specifically, the number of seconds since 00:00:00 GMT Jan 1, 1970.</p> <p>The function can also convert just the date ("<i>dd mmm yy</i>"), in which case it returns the number of seconds since 00:00:00 GMT Jan 1, 1970 from midnight of the entered day. It will also convert a time without the date ("<i>hh:mm:ss</i>"), in which case it returns the number of seconds that have passed since midnight of that day.</p> <p>If the format is incorrect or can't be interpreted, then <b>Time.DateSec</b> will return -1 and <b>Error.Last</b> and <b>Error.LastMsg</b> will be set to the appropriate error.</p> |
| Example     | <p>This program gives the number of seconds since 00:00:00 GMT Jan 1, 1970.</p> <pre> <b>var</b> theDateTime, theDate, theTime : <b>string</b> theDateTime := <b>Time.Date</b> theDate := theDateTime (1 .. 9) theTime := theDateTime (11 .. *) <b>put</b> "The number of seconds from 00:00:00 GMT Jan 1, 1970",     "from midnight ", theDate, "is ", <b>Time.DateSec</b> (theDate) <b>put</b> "The number of seconds from midnight to ", theTime "is ",     <b>Time.DateSec</b> (theTime) <b>put</b> "The number of seconds from 00:00:00 GMT Jan 1, 1970",     "from ", theDateTime, "is ", <b>Time.DateSec</b> (theDateTime) </pre>                                                                            |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Time.DateSec</b> , not by calling <b>DateSec</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| See also    | <b>Time.Sec</b> , <b>Time.Date</b> and <b>Time.SecDate</b> functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

# Time.Delay

All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.Delay</b> ( <i>duration</i> : int )                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | The <b>Time.Delay</b> procedure is used to cause the program to pause for a given time. The time duration is in milliseconds.                                                                                                                                                                                                                                                                                                                                                               |
| Example     | This program prints the integers 1 to 10 with a second delay between each.<br><pre>for i : 1 .. 10   put i   Time.Delay ( 1000 )  % Pause for 1 second end for</pre>                                                                                                                                                                                                                                                                                                                        |
| Details     | <p>On IBM PC compatibles, the hardware resolution of duration is in units of 55 milliseconds. For example, <b>Time.Delay</b> (500) will delay the program by about half a second, but may be off by as much as 55 milliseconds.</p> <p>On Apple Macintoshes, the hardware resolution of duration is in units of 17 milliseconds (1/60th of a second). For example, <b>Time.Delay</b> (500) will delay the program by about half a second, but may be off by as much as 17 milliseconds.</p> |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Time.Delay</b> , not by calling <b>Delay</b> .                                                                                                                                                                                                                                                                                                                                                          |
| See also    | <b>Time.Elapsed</b> and <b>Time.ElapsedCPU</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                            |

# Time.Elapsed

All

|             |                                                                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.Elapsed</b> : int                                                                                                                                                        |
| Description | The <b>Time.Elapsed</b> function returns the amount of time since a program (process) started running. The number of milliseconds since the program started running is returned. |
| Example     | This program tells you how much time it has used.                                                                                                                                |

```

var timeRunning : int
timeRunning := Time.Elapsed
 put "This program has run ", timeRunning, " milliseconds"

```

|          |                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | <p>On IBM PC compatibles, this is the total time since the Turing system was started up. The hardware resolution of duration is in units of 55 milliseconds. For example, <b>Time.Elapsed</b> may be off by as much as 55 milliseconds.</p> <p>On Apple Macintoshes, this is the total time since the machine was turned on. The hardware resolution of duration is in units of 17 milliseconds (1/60-th of a second).</p> |
| Status   | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Time.Elapsed</b>, not by calling <b>Elapsed</b>.</p>                                                                                                                                                                                                                                                                            |
| See also | <b>Time.ElapsedCPU</b> and <b>Time.Delay</b> subprograms.                                                                                                                                                                                                                                                                                                                                                                  |

## Time.ElapsedCPU All

|             |                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.ElapsedCPU : int</b>                                                                                                                                                                                                                                                                                                                                                        |
| Description | <p>The <b>Time.ElapsedCPU</b> function is used on a multitasking system such as UNIX to determine the amount of time that has been used by this program (process). The number of central processor milliseconds assigned to this program is returned. This is of little use on a personal computer, where <b>Time.ElapsedCPU</b> returns the same value as <b>Time.Elapsed</b>.</p> |
| Example     | <p>On a UNIX system, this program tells you how much time it has used.</p> <pre> var timeUsed : int timeUsed := <b>Time.ElapsedCPU</b> put "This program has used ", timeUsed,     " milliseconds of CPU time" </pre>                                                                                                                                                               |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Time.ElapsedCPU</b>, not by calling <b>ElapsedCPU</b>.</p>                                                                                                                                                                                                                               |
| See also    | <b>Time.Elapsed</b> and <b>Time.Delay</b> subprograms.                                                                                                                                                                                                                                                                                                                              |

## Time.PartsSec All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.PartsSec</b> ( <i>year, month, day, hour, minute, second</i> : <b>int</b> ) : <b>int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Description | <p>The <b>Time.PartsSec</b> function is used to convert the numeric parts of a date (specifically the year, month, day, hour, minute and second) into the number of seconds since 00:00:00 GMT Jan 1, 1970 and the date specified by the parts.</p> <p>The function can also convert a time without a date (year, month and day are all 0), in which case it returns the number of seconds that have passed since midnight of the current day.</p> <p>If the numbers don't make any sense or can't be interpreted, then <b>Time.PartsSec</b> will return -1 and <b>Error.Last</b> and <b>Error.LastMsg</b> will be set to the appropriate error.</p> |
| Example     | <p>This program gives the number of seconds between 00:00:00 GMT Jan 1, 1970 and 9:27 in the morning, Christmas Day, 1989).</p> <pre>put "The number of seconds from 00:00:00 GMT Jan 1, 1970",<br/>    "is ", Time.PartsSec (1989, 12, 25, 9, 27, 0)</pre>                                                                                                                                                                                                                                                                                                                                                                                          |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Time.PartsSec</b>, not by calling <b>PartsSec</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| See also    | <b>Time.SecParts</b> , <b>Time.Date</b> and <b>Time.Sec</b> functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## Time.Sec All

|             |                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.Sec</b> : <b>int</b>                                                                                                                                                             |
| Description | <p>The <b>Time.Sec</b> function returns the current date and time as a number. The returned integer is the time in seconds since 00:00:00 GMT (Greenwich Mean Time) January 1, 1970.</p> |
| Example     | <p>This program tells you how many seconds since 1970.</p> <pre>put "The number of seconds since 1970 is ", Time.Sec</pre>                                                               |



|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | <p>Be warned that on some computers, such as IBM PC compatibles or Apple Macintoshes, the date may not be set correctly in the operating system; in that case, the <b>Time.Date</b> procedure will give incorrect results.</p> <p>The string form of the date can be converted to a numeric form for comparison purposes using the <b>Time.DateSec</b> function. The numeric form can be converted to a string using the <b>Time.SecDate</b> function. The numeric form of the time can be obtained using the <b>Time.Sec</b> function.</p> |
| Status   | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Time.Sec</b>, not by calling <b>Sec</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                     |
| See also | <b>Time.Date</b> , <b>Time.DateSec</b> and <b>Time.SecDate</b> functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

## Time.SecDate All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.SecDate</b> ( <i>timeInSecs</i> : <b>int</b> ) : <b>string</b>                                                                                                                                                                                                                                                                                                                                          |
| Description | <p>The <b>Time.SecDate</b> function is used to convert the number of seconds since 00:00:00 GMT Jan 1, 1970 into a date and time string.</p> <p>If <i>timeInSecs</i> is incorrect or can't be interpreted, then <b>Time.SecDate</b> will return the empty string and <b>Error.Last</b> and <b>Error.LastMsg</b> will be set to the appropriate error.</p>                                                       |
| Example     | <p>This program gives the number of seconds since 00:00:00 GMT Jan 1, 1970 and the date in string form.</p> <pre> <b>var</b> timeInSecs : <b>int</b> := <b>Time.Sec</b> <b>var</b> theDateTime: <b>string</b> theDateTime := <b>Time.SecDate</b> (timeInSecs) <b>put</b> "The number of seconds since 1970 is ", timeInSecs            <b>put</b> "Greetings!! The date and time today is ", theDateTime </pre> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Time.SecDate</b>, not by calling <b>SecDate</b>.</p>                                                                                                                                                                                                                                                                 |
| See also    | <b>Time.Sec</b> , <b>Time.Date</b> and <b>Time.DateSec</b> functions.                                                                                                                                                                                                                                                                                                                                           |

# Time.SecParts All

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Time.SecParts</b> ( <i>sec : int, var year, month, day, dayOfWeek, hour, minute, second : int</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Description | <p>The <b>Time.SecParts</b> function is used to convert a single number form of the time (the number of seconds since 00:00:00 GMT Jan 1, 1970) into a date with numeric component parts.</p> <p>The <i>dayOfWeek</i> parameter is 1 for Monday, 2 for Tuesday through 7 for Sunday.</p> <p>If the <i>sec</i> parameter doesn't make any sense or can't be interpreted, then <b>Time.PartsSec</b> will set all the <b>var</b> parameters to -1 and <b>Error.Last</b> and <b>Error.LastMsg</b> will be set to the appropriate error.</p> |
| Example     | <p>This program returns the current day of the week.</p> <pre>var year, month, day, dayOfWeek, hour, minute, second : int Time.SecParts (Time.Sec, year, month, day, dayOfWeek,                hour, minute, second ) var days : array 1 .. 7 of string (10) := init ("Monday", "Tuesday",         "Wednesday", "Thursday", "Friday", "Saturday", "Sunday") put "The current day of the week is ", days (dayOfWeek)</pre>                                                                                                               |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Time.SecParts</b> , not by calling <b>SecParts</b> .</p>                                                                                                                                                                                                                                                                                                                                                                                     |
| See also    | <b>Time.PartsSec</b> , <b>Time.Date</b> and <b>Time.Sec</b> functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

## time     time of day as a string procedure

|        |                                                     |
|--------|-----------------------------------------------------|
| Syntax | <b>time</b> ( <b>var</b> <i>t</i> : <b>string</b> ) |
|--------|-----------------------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | The <b>time</b> statement is used to determine the current time of day. Variable <i>t</i> is assigned a string in the format " <i>hh:mm:ss</i> ". For example, if the time is two minutes and 47 seconds after nine A.M., <i>t</i> will be set to "09:02:47". Twenty-four hour time is used. For example, eleven thirty P.M. gives the string "23:30:00". |
| Example     | This program greets you and tells you the time of day.<br><pre> <b>var</b> timeOfDay : <b>string</b> <b>time</b> ( timeOfDay )     <b>put</b> "Greetings!! The time is ", timeOfDay </pre>                                                                                                                                                                |
| Details     | Be warned that on some computers such as IBM PC compatibles or Apple Macintoshes, the time may not be set correctly in the operating system. In this case, the <b>time</b> procedure will give incorrect results.                                                                                                                                         |
| See also    | <b>delay</b> , <b>clock</b> , <b>sysclock</b> , <b>wallclock</b> and <b>date</b> statements.<br>See also predefined unit <b>Time</b> .                                                                                                                                                                                                                    |

## token in input

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | A <i>token</i> is essentially a word, a number or a special symbol such as :=. In a Turing program there are four kinds of tokens: keywords such as <b>get</b> , identifiers such as <i>incomeTax</i> , operators and special symbols, such as + and :=, and explicit constants, such as 1.5 and "Hello". Some keywords, such as <b>index</b> , are reserved and cannot be used in programs to name variables, procedures, etc.<br>A <b>get</b> statement, such as<br><pre> <b>get</b> incomeTax </pre> uses <i>token-oriented</i> input. This means that white space (blanks, tabs, etc.) is skipped before reading the input item and after the item (up to the beginning of the next line). See the <b>get</b> statement for details. |
| Example     | In this example, the tokens are <b>var</b> , <i>x</i> , :, <b>real</b> , <i>x</i> , := and 9.84.<br><pre> <b>var</b> x : <b>real</b> x := 9.84 </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## true    boolean value (not false)

|             |                                                                                                                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>true</b>                                                                                                                                                                                     |
| Description | A <b>boolean</b> (true/false) variable can be either <b>true</b> or <b>false</b> (see <b>boolean</b> type).                                                                                     |
| Example     | <pre>var passed : boolean := true var mark : int for i : 1 .. 10   get mark   passed := passed and mark &gt;= 60 end for if passed = true then   put "You passed all ten subjects" end if</pre> |
| Details     | The line <b>if</b> <i>passed</i> = <b>true</b> <b>then</b> can be simplified to <b>if</b> <i>passed</i> <b>then</b> with no change to the meaning of the program.                               |

## TypeConv All

|              |                                                                                                                                                                                                                                                                                                                                                                                                                      |                                      |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that convert between different Turing standard types. There are also six routines that are part of the language, rather than part of the unit, but are conceptually part of this unit.</p> <p>All routines in the <b>TypeConv</b> unit are exported unqualified.</p> <p>Description of the routines in the <b>TypeConv</b> module can be found in this chapter.</p> |                                      |
| Entry Points | <b>intreal</b>                                                                                                                                                                                                                                                                                                                                                                                                       | Converts an integer to a real.       |
|              | <b>intstr</b> *                                                                                                                                                                                                                                                                                                                                                                                                      | Converts an integer to a string.     |
|              | <b>natreal</b>                                                                                                                                                                                                                                                                                                                                                                                                       | Converts a natural number to a real. |

|              |                   |                                                                                                                                |
|--------------|-------------------|--------------------------------------------------------------------------------------------------------------------------------|
|              | <b>natstr</b> *   | Converts a natural number to a string.                                                                                         |
|              | <b>round</b>      | Converts a real to an integer (rounding).                                                                                      |
|              | <b>floor</b>      | Converts a real to an integer (round down).                                                                                    |
|              | <b>ceil</b>       | Converts a real to an integer (round up).                                                                                      |
|              | <b>realstr</b>    | Converts a real to a string.                                                                                                   |
|              | <b>erealstr</b>   | Converts a real to a string (exponential notation).                                                                            |
|              | <b>frealstr</b>   | Converts a real to a string (no exponent).                                                                                     |
|              | <b>strint</b> *   | Converts a string to an integer.                                                                                               |
|              | <b>strintok</b> * | Returns whether a string can legally be converted to an integer.                                                               |
|              | <b>strnat</b> *   | Converts a string to a natural number.                                                                                         |
|              | <b>strnatok</b> * | Returns whether a string can legally be converted to a natural number.                                                         |
|              | <b>streal</b>     | Converts a string to a real.                                                                                                   |
|              | <b>strealok</b>   | Returns whether a string can legally be converted to a real.                                                                   |
| <b>ord</b> * | <b>chr</b> *      | Returns the ASCII value of a specified string of length one.<br>Returns a string of length one with the ASCII value specified. |

\* Part of the language, conceptually part of the **TypeConv** unit.

## type declaration

|             |                                                                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>typeDeclaration</i> is one of:</p> <p>(a) <b>type</b> <i>id</i> : <i>typeSpec</i></p> <p>(b) <b>type</b> <i>id</i> : <b>forward</b></p> |
| Description | A type declaration gives a name to a type. This name can be used in place of the type.                                                          |

### Example

```
type nameType : string (30)
type range : 0 .. 150
type entry :
 record
 name : nameType
 age : int
 end record
```

### Details

The keyword **pervasive** can be inserted just after **type**. When this is done, the type is visible inside all subconstructs of the type scope. Without **pervasive**, the type is not visible inside modules, monitors and classes unless explicitly imported. Pervasive types need not be imported. You can abbreviate **pervasive** as an asterisk (\*).

A **forward** type allows pointers to be declared to the **type** before the type is *resolved*. To resolve a **type**, you must follow a **forward** with a declaration of the same name and in the same scope. This type declaration must include a *typeSpec*.

## typeSpec    type specification

### Syntax

A *typeSpec* (type specification) is one of:

- (a)    **int**
- (b)    **real**
- (c)    **boolean**
- (d)    *stringType*            % Example: **string** ( 20 )
- (e)    *subrangeType*        % Example: 1 .. 150
- (f)    *enumeratedType*    % Example: **enum** ( red, green, blue )
- (g)    *arrayType*            % Example: **array** 1 .. 150 **of** *real*
- (h)    *setType*              % Example: **set of** 1 .. 10
- (i)    *recordType*          % Example: **record** ... **end record**
- (j)    *unionType*          % Example: **union** ... **end union**
- (k)    *pointerType*        % Example: **pointer to** collectionVar
- (l)    *namedType*          % Example: colorRange
- (m)    **nat**                  % natural number
- (n)    **intn**                % n-byte integer (n=1, 2, 4)

- (o) **nat***n* % *n*-byte natural (*n*= 1, 2, 4)
- (p) **real***n* % *n*-byte real (*n*=4, 8)
- (q) **char** % single character
- (r) **char**(*n*) % *n* characters
- (s) *subprogramType*

**Description** A type specification determines the allowed values for a variable or constant. For example, if variable *x* is an integer (its *typeSpec* is **int**), the possible values for *x* are numbers such as -15, 0, 3 and 348207. If *x* is a real number (its *typeSpec* is **real**), then its possible values include 7.8, -35.0, and 15e12. If *x* is a **boolean**, its possible values are **true** and **false**. If *x* is a **string**, its possible values include *Hello* and *Good-bye*.

**Example**

```

var numberOfSides : int
var x, y : real
type range : 0 .. 150 % The typeSpec here is 0 .. 150
type entry : % Here is a record typeSpec
 record
 name : string (25)
 age : range
 end record

```

## unchecked compiler directive

Dangerous

**Description** OOT adds the concept of "unchecked" to Turing. Here, you can request that certain run time tests, which take place by default, can be eliminated. This makes the program more efficient at the risk of unreliability.

**Example** Declare *p* to be an unchecked pointer to an integer (see **pointers** for details). Pointer *p* will be dangerous to use, because the run time system will not check to see if it actually locates an integer, as opposed to arbitrary computer memory. In other words, unchecked pointers are like C language pointers.

```
var p : unchecked ^ int
```

**Example** Declare *C* to be an unchecked collection of records of type *R* (see **collections** for details). Pointers to *C* will be unchecked.

```
var C : unchecked collection of R
```

**Example** Remove checking from the body of a loop.

```
for i : 1 .. 500
```

```

unchecked
if a (i) = key then
 exit
end if
end for

```

Details            In the above example, the **unchecked** keyword requests that all checking, in particular, array bounds checking for array *a*, are to be omitted. The disabling lasts from the occurrence of the keyword **unchecked** to the end of the surrounding construct, in this case, until **end for**. In a similar way, the checked keyword will request that checking be re-enabled from the occurrence of **checked** to the end of the surrounding construct.

In the current (1999) implementation, the use of **unchecked** to turn off checking in a block of statements is ignored. In general, an implementation may choose to ignore requests to disable checking.

## union    type

Syntax            A *unionType* is:

```

union [id] : indexType of
 label labelExpn { , labelExpn } :
 { id { , id } : typeSpec }
 { label labelExpn { , labelExpn } :
 { id { , id } : typeSpec } }
 [label : { id { , id } : typeSpec }]
end union

```

Description       A union type (also called a variant record) is like a record in which there is a run time choice among sets of accessible fields. This choice is made by the **tag** statement, which deletes the current set of fields and activates a new set.

Example            This union type keeps track of various information about a vehicle, depending on the kind of vehicle.

```

const passenger := 0
const farm := 1
const recreational := 2

type vehicleInfo :
 union kind : passenger .. recreational of
 label passenger :
 cylinders : 1..16
 label farm :
 farmClass : string (10)
 label : % No fields for "otherwise" clause
 end union
end type

```



```

 end union
 var v : vehicleInfo
 ...
 tag v, passenger % Activate passenger part v.cylinders := 6

```

Details        The optional identifier following the keyword **union** is the name of the *tag* of the union type. If the identifier is omitted, the tag is still considered to exist, although its value cannot be accessed. The tag must be of an index type, for example 1..7. You should limit the range of this index type, as the compiler may have a limit (at least 255) on the maximum range it can handle.

Each *labelExprn* must be known at compile time and must lie within the range of the tag's type. The fields, including the tag, of a union value are referenced using the dot operator, as in *v.cylinders* and these can be used as variables or constants. A field can be accessed only when the tag matches one of the label expressions corresponding to the field. The tag can be changed by the **tag** statement but it cannot be assigned to, passed to a **var** parameter, or bound to using **var**.

In a union, *id*'s of fields, including the tag, must be distinct. However, these need not be distinct from identifiers outside the union. Unions can be assigned as a whole (to unions of an equivalent type), but they cannot be compared. A semicolon can optionally follow each *typeSpec*.

Any array contained in a union must have bounds that are known at compile time.

The notation *->* can be used to access union fields. For example, if *p* is a pointer to *vehicleRecord*, *p->farmClass* locates the *farmClass* field.

See also        **pointer**.

## unit        file containing module, monitor, or class

Syntax        A *compilationUnit* is one of:

- (a)    [ *importList* ] *mainProgram*
- (b)    **unit** *moduleDeclaration*
- (c)    **unit** *monitorDeclaration*
- (d)    **unit** *classDeclaration*

**Description**      A program can be divided up into units, each in a separate file. All of these files except the main program begin with the keyword **unit**. The unit contains the main program, a module, a monitor or a class.

**Example**            Here is stack module that is separated out into a file whose name is *stack*.

```
unit % The keyword unit begins each separate file
module stack
 export push, pop
```

```
 var top : int := 0
 var contents : array 1 .. 100 of int
```

```
 procedure push (i : int)
 top += 1
 contents (top) := i
 end push
```

```
 procedure pop (i : int)
 i := contents (top)
 top -= 1
 end pop
```

```
 end stack
```

The main program, which is in another file, gains access to the stack by importing it. Here is the main program:

```
import var stack % Use the stack
var n : int
...
stack . push (n)
...
 stack . pop (n)
```

**Details**            In this example, the keyword **var** in the import list is required because the main program causes a change in the stack, by calling *push* and *pop*. The import lists of units that are modules, monitors and classes are used to gain access to further units.

If the stack were in a file with a different name, say *stk.t*, the import list would be rewritten to use an **in** clause, as follows:

```
 import var stack in "stk.t"
```

A mainProgram is simply a program. See **program**.

**See also**            **module**, **monitor** and **class**. See also **export** list, **import** list, **inherit** list, **implement** list and **implement by** list.

## unqualified export

|             |                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | When an identifier $x$ is exported from a module, monitor or class $M$ using the keyword <b>unqualified</b> , it can be used outside of $M$ without the qualification " $M$ ". In other words, outside of $M$ , it can be referred to as simply $x$ . The keyword <b>unqualified</b> can be written in its short form as $\sim$ , which is pronounced "not dot", as in:<br><b>export</b> $\sim$ . $x$ |
| See also    | <b>export</b> list.                                                                                                                                                                                                                                                                                                                                                                                   |

## upper bound

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>upper</b> ( <i>reference</i> [ , <i>dimension</i> ] ) : <b>int</b>                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | The <b>upper</b> attribute is used to find the upper bound of an array, string, <b>char</b> ( $n$ ) or non-opaque subrange type. (See <b>lower</b> for finding the lower bound.)                                                                                                                                                                                                                                                                      |
| Example     | In a procedure, see if the bound of array parameter $a$ is large enough that it can be subscripted by $i$ . If it is large enough, it is set $a(i)$ to zero.<br><pre><b>procedure</b> test ( <b>var</b> <math>a</math> : <b>array</b> 1 .. * <b>of</b> <b>real</b> )<br/>  <b>if</b> <math>i \leq</math> <b>upper</b> ( <math>a</math> ) <b>then</b><br/>    <math>a</math> ( <math>i</math> ) := 0.0<br/>  <b>end if</b><br/>  <b>end</b> test</pre> |
| Details     | In a similar way, if $s$ is a string, its upper bound (not length!) is given by <b>upper</b> ( $s$ ). If an array has more than one dimension, as in <b>var</b> $b$ : <b>array</b> 1..10, 1 .. 60 <b>of</b> <b>int</b> , you must specify the dimension. For example, <b>upper</b> ( $b$ , 2) returns 60.                                                                                                                                             |

# var declaration

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <p>A <i>variableDeclaration</i> is one of:</p> <p>(a) <b>var</b> <i>id</i> { ,<i>id</i> } [ :<i>typeSpec</i> ]<br/> [ :=<i>initializingValue</i> ]</p> <p>(b) <i>collectionDeclaration</i></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Description | <p>A variable declaration creates a new variable (or variables). Only form (a) will be explained here. See <i>collectionDeclaration</i> for explanation of form (b). The <i>typeSpec</i> of form (a) can be omitted only if the initializing value is present.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Example     | <pre> <b>var</b> j, k : <b>int</b> := 1           % j and k are assigned value 1 <b>var</b> t := "Sample"           % The type of t is <b>string</b> <b>var</b> v : <b>array</b> 1 .. 3 <b>of string</b> ( 6 ) :=                                 <b>init</b> ( "George", "Fred", "Alice" ) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Details     | <p>The initializing value, if present, must be an expression or else a list of items separated by commas inside <b>init</b> ( ... ). The syntax of <i>initializingValue</i> is one of:</p> <p>(a) expn</p> <p>(b) <b>init</b> ( <i>initializingValue</i> { , <i>initializingValue</i> } )</p> <p>Each <b>init</b> ( ... ) corresponds to an array, record or union value that is being initialized. These must be nested for initialization of nested types.</p> <p>If the <i>typeSpec</i> is omitted, the variable's type is taken to be the (root) type of the initializing expression, for example, <b>int</b> or <b>string</b>. The <i>typeSpec</i> cannot be omitted for dynamic arrays or when the initializing value is of the form <b>init</b> ( ... ). The values inside <b>init</b> ( ... ) must be known at compile time.</p> <p>The keyword <b>pervasive</b> can be inserted just after <b>var</b>. When this is done, the variable is visible inside all subconstructs of the variable's scope. Without <b>pervasive</b>, the variable is not visible inside modules unless explicitly imported. Pervasive variables need not be imported. You can abbreviate <b>pervasive</b> as an asterisk (*).</p> <p>OOT extends Turing in the following way. OOT changes form (a) to allow the optional use of the <b>register</b> keyword to request that the variable be placed in a machine register. The OOT syntax for form (a) is actually:</p> <pre> <b>var</b> [<b>pervasive</b>] [<b>register</b>] <i>id</i> { , <i>id</i> } [ : <i>typeSpec</i> ] [ := <i>initializingValue</i> ] </pre> <p>In the current (1994) OOT implementation, programs are run interpretively using pseudo-code, which has no machine registers, and the <b>register</b> keyword is ignored. See <b>register</b> for restrictions on the use of register variables.</p> |

See also **collection**, **bind**, **procedure** and **function** declarations, parameter declarations, **export** lists and **import** lists for other uses of the keyword **var**.

## variableReference    use of a variable

Syntax            A *variableReference* is:

*variableId* { *componentSelector* }

Description      In a Turing program, a variable is declared and given a name (*variableId*) and then used. Each use is called a *variable reference*.  
If the variable is an array, collection, record or union, its parts (*components*) can be selected using subscripts and field names (using *componentSelectors*). The form of a *componentSelector* is one of:\

(a)            ( *expn* {, *expn*} )

(b)            . *fieldId*

Form (a) is used for subscripting (indexing) arrays and collections. The number of array subscripts must be the same as in the array's declaration. A collection has exactly one subscript, which must be a pointer to the collection. Form (b) is used for selecting a field of a record or union.

Example            Following the declarations of *k*, *a* and *r*, each of *k*, *a* (*k*) and *r.name* are variable references.

```
var k : int
var a : array 1 .. 100 of real
var r :
 record
 name : string (20)
 phone : string (8)
 end record
...
k := 5
a (k) := 3.14159
r . name := "Steve Cook"
```

Details            A variable reference can contain more than one component selector, for example, when the variable is an array of records. For an example, see the **record** type. See also *constantReference* and **var** declaration.

# View

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that deal with the current output surface. This can be a screen (in the DOS version of OOT) or a window (in the X Windows, MS Windows or Macintosh version).</p> <p>All routines in the <b>View</b> unit are exported qualified (and thus must be prefaced with "<b>View.</b>") with the exception of <b>maxx</b>, <b>maxy</b>, <b>maxcolor</b> and <b>maxcolour</b> which are exported unqualified.</p> |                                                                         |
| Entry Points | <b>maxx</b>                                                                                                                                                                                                                                                                                                                                                                                                                                               | Returns the maximum x coordinate (width - 1) (exported unqualified).    |
|              | <b>maxy</b>                                                                                                                                                                                                                                                                                                                                                                                                                                               | Returns the maximum y coordinate (height - 1) (exported unqualified).   |
|              | <b>maxcolor</b>                                                                                                                                                                                                                                                                                                                                                                                                                                           | Returns the maximum color number (# colors - 1) (exported unqualified). |
|              | <b>maxcolour</b>                                                                                                                                                                                                                                                                                                                                                                                                                                          | Returns the maximum color number (# colors - 1) (exported unqualified). |
|              | <b>Set</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                | Changes the configuration of the output surface.                        |
|              | <b>ClipSet</b>                                                                                                                                                                                                                                                                                                                                                                                                                                            | Clips output to a specified rectangle.                                  |
|              | <b>ClipAdd</b>                                                                                                                                                                                                                                                                                                                                                                                                                                            | Adds another specified rectangle to the clipping region.                |
|              | <b>ClipOff</b>                                                                                                                                                                                                                                                                                                                                                                                                                                            | Stops all clipping.                                                     |
|              | <b>WhatDotColor</b>                                                                                                                                                                                                                                                                                                                                                                                                                                       | Gets the color of the pixel at a specified location.                    |
|              | <b>WhatDotColour</b>                                                                                                                                                                                                                                                                                                                                                                                                                                      | Gets the color of the pixel at a specified location.                    |
|              | <b>GetActivePage</b>                                                                                                                                                                                                                                                                                                                                                                                                                                      | Returns the currently active graphics page.                             |
|              | <b>SetActivePage</b>                                                                                                                                                                                                                                                                                                                                                                                                                                      | Sets the currently active graphics page.                                |
|              | <b>GetVisiblePage</b>                                                                                                                                                                                                                                                                                                                                                                                                                                     | Returns the currently visible graphics page.                            |
|              | <b>SetVisiblePage</b>                                                                                                                                                                                                                                                                                                                                                                                                                                     | Sets the currently visible graphics page.                               |

# View.ClipAdd



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>View.ClipAdd</b> ( <i>x1, y1, x2, y2</i> : int)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Description | <p>The <b>View.ClipAdd</b> procedure adds another rectangle specified by (<i>x1, y1</i>) - (<i>x2, y2</i>) to the clipping region. This only works on systems that support complex clipping regions. If no clipping region has been specified, then the rectangle becomes the complete clipping region.</p> <p>A clipping region is the region that the output will appear in. If the rectangle is specified as the clipping region, any drawing done outside the rectangle will not appear.</p> <p>To set the initial clipping, or remove the old region and replace it with a new one, use <b>View.ClipSet</b>. To set the clipping region back to the entire screen or window, use <b>View.ClipOff</b>.</p> <p>These commands only work in "graphics" mode.</p>                   |
| Example     | <p>This program sets the clipping region to five rectangles and then draws random circles. The circles will only appear (or partially appear) in the rectangles.</p> <pre> const maxx13 : int := maxx div 3 const maxx23 : int := 2 * maxx div 3 const maxy13 : int := maxy div 3 const maxy23 : int := 2 * maxy div 3 View.ClipSet (0, 0, maxx13, maxy13) View.ClipAdd (maxx23, 0, maxx, maxx13) View.ClipAdd (maxx13, maxy13, maxx23, maxy23) View.ClipAdd (0, maxy23, maxx13, maxy) View.ClipAdd (maxx23, maxy23, maxx, maxy)  % Draw the random ovals in the box var x, y, clr : int loop   x := Rand.Int (0, maxx)      % Random x   y := Rand.Int (0, maxy)      % Random y   clr := Rand.Int (0, maxcolor) % Random color   Draw.FillOval (x, y, 30, 30, clr) end loop </pre> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>View.ClipAdd</b>, not by calling <b>ClipAdd</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| See also    | <b>View.ClipSet</b> and <b>View.ClipOff</b> functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

# View.ClipOff



|             |                                                                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>View.ClipOff</b>                                                                                                                                                                                                                                        |
| Description | <p>The <b>View.ClipOff</b> procedure turns off clipping. This means that any drawing commands can appear on the entire drawing surface (the screen or the window, depending on the system).</p> <p>These commands only work in "<i>graphics</i>" mode.</p> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>View.ClipOff</b>, not by calling <b>ClipOff</b>.</p>                                                                                                            |
| See also    | <b>View.ClipAdd</b> and <b>View.ClipSet</b> functions.                                                                                                                                                                                                     |

# View.ClipSet



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>View.ClipSet</b> ( <i>x1, y1, x2, y2</i> : int)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Description | <p>The <b>View.ClipSet</b> procedure sets the clipping region to the rectangle specified by (<i>x1, y1</i>) - (<i>x2, y2</i>). If a clipping region already exist, it is replaced by the specified rectangle.</p> <p>A clipping region is the region in which the output will appear. If the rectangle is specified as the clipping region, any drawing done outside the rectangle will not appear.</p> <p>To set the initial clipping, or remove the old region and replace it with a new one, use <b>View.ClipSet</b>. To set the clipping region back to the entire screen or window, use <b>View.ClipOff</b>.</p> <p>These commands only work in "<i>graphics</i>" mode.</p> |
| Example     | <p>This program sets the clipping region to five rectangles and then draws random circles. The circles will only appear (or partially appear) in the rectangles.</p> <pre>const maxx13 : int := maxx div 3 const maxx23 : int := 2 * maxx div 3 const maxy13 : int := maxy div 3</pre>                                                                                                                                                                                                                                                                                                                                                                                           |



```

const maxy23 : int := 2 * maxy div 3
View.ClipSet (0, 0, maxx13, maxy13)
View.ClipAdd (maxx23, 0, maxx, maxy13)
View.ClipAdd (maxx13, maxy13, maxx23, maxy23)
View.ClipAdd (0, maxy23, maxx13, maxy)
View.ClipAdd (maxx23, maxy23, maxx, maxy)

% Draw the random ovals in the box
var x, y, clr : int
loop
 x := Rand.Int (0, maxx) % Random x
 y := Rand.Int (0, maxy) % Random y
 clr := Rand.Int (0, maxcolor) % Random color
 Draw.FillOval (x, y, 30, 30, clr)
end loop

```

Status            Exported qualified.  
                     This means that you can only call the function by calling **View.ClipSet**, not  
                     by calling **ClipSet**.

See also           **View.ClipAdd** and **View.ClipOff** functions.

## View.GetActivePage



Syntax            **View.GetActivePage : int**

Description       In certain graphics modes under DOS OOT, it is possible to have multiple  
                     graphics pages. This means it is possible to draw to one graphics page  
                     while another is visible. You can use **View.SetActivePage** to specify which  
                     graphics page is being drawn upon and use **View.SetVisiblePage** to  
                     specify the graphics page that is to be visible.

                    By drawing on a different graphics page from the currently visible page, it  
                     is possible to avoid the flicker that might occur while the image is being  
                     composed, especially if it is composed of several different parts.

                    The general method is to make the second page the active page, draw the  
                     image upon it, then make the second page visible. Then make the first page  
                     active and draw upon it while the user is looking at the second graphics  
                     page.

**View.GetActivePage** returns the currently active graphics page. The  
                     default page is 1.

|          |                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | <p>You must use <b>Config.Display</b> in order to determine the number of graphics modes available. The <i>cdMaxNumPages</i> parameter will return the number of graphics pages available.</p> <p>At the time of writing, only “ega” graphics mode had more than one graphics page. This may change with future releases. Consult the release notes for any changes that may have occurred.</p> |
| Status   | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>View.GetActivePage</b>, not by calling <b>GetActivePage</b>.</p>                                                                                                                                                                                                                                     |
| See also | <p><b>View.SetActivePage</b>, <b>View.GetVisiblePage</b> and <b>View.SetVisiblePage</b> for routines to switch active and visible graphics pages.</p>                                                                                                                                                                                                                                           |

## View.GetVisiblePage

Win DOS Mac  
  
 X TOOT Term

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>View.GetVisiblePage : int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | <p>In certain graphics modes under DOS OOT, it is possible to have multiple graphics pages. This means it is possible to draw to one graphics page while another is visible. You can use <b>View.SetActivePage</b> to specify which graphics page is being drawn upon and use <b>View.SetVisiblePage</b> to specify the graphics page that is to be visible.</p> <p>By drawing on a different graphics page from the currently visible page, it is possible to avoid the flicker that might occur while the image is being composed, especially if it is composed of several different parts.</p> <p>The general method is to make the second page the active page, draw the image upon it, then make the second page visible. Then make the first page active and draw upon it while the user is looking at the second graphics page.</p> <p><b>View.GetVisiblePage</b> returns the currently visible graphics page. The default page is 1.</p> |
| Details     | <p>You must use <b>Config.Display</b> in order to determine the number of graphics modes available. The <i>cdMaxNumPages</i> parameter will return the number of graphics pages available.</p> <p>At the time of writing, only “ega” graphics mode had more than one graphics page. This may change with future releases. Consult the release notes for any changes that may have occurred.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

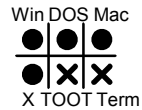
|          |                                                                                                                                                      |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>View.GetVisiblePage</b> , not by calling <b>GetVisiblePage</b> . |
| See also | <b>View.SetVisiblePage</b> , <b>View.GetVisiblePage</b> and <b>View.SetVisiblePage</b> for routines to switch active and visible graphics pages.     |

## View.maxcolor



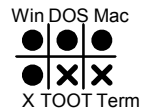
|             |                                                                                                                                                                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>View.maxcolor : int</b>                                                                                                                                                                                                                                                                                                               |
| Description | The <b>maxcolor</b> function is used to determine the maximum color number for the current mode of the screen. The alternate spelling is <b>maxcolour</b> .                                                                                                                                                                              |
| Example     | This program outputs the maximum color number.<br><pre>setscreen ("graphics") ...       put "The maximum color number is ", View.maxcolor</pre>                                                                                                                                                                                          |
| Details     | The screen should be in a <i>"screen"</i> or <i>"graphics"</i> mode. If it is not, it will automatically be set to <i>"screen"</i> mode. See <b>View.Set</b> for details.<br>For IBM PC compatibles in <i>"screen"</i> mode, <b>maxcolor</b> = 15. For the default IBM PC compatible <i>"graphics"</i> mode (VGA), <b>maxcolor</b> = 15. |
| Details     | <b>View.maxcolor</b> is identical to <b>RGB.maxcolor</b> . It is placed here for consistency with other screen information routines.                                                                                                                                                                                                     |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>View.maxcolor</b> . Note that <b>RGB.maxcolor</b> is exported unqualified, so that one can call <b>maxcolor</b> .                                                                                                                                    |
| See also    | <b>Draw.Dot</b> for examples of the use of <b>maxcolor</b> . See the <b>Text.Color</b> procedure which is used for setting the currently-active color.                                                                                                                                                                                   |

# View.maxx



|             |                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>maxx : int</b>                                                                                                                                                                                                        |
| Description | The <b>maxx</b> function is used to determine the maximum value of x for the current graphics mode.                                                                                                                      |
| Example     | This program outputs the maximum x value.<br><pre>setscreen ("graphics")<br/>...<br/>put "The maximum x value is ", maxx</pre>                                                                                           |
| Details     | The screen should be in a "graphics" mode. If it is not, it will automatically be set to "graphics" mode. See <b>setscreen</b> for details.<br>For the default IBM PC compatible graphics mode (CGA), <b>maxx</b> = 319. |
| Status      | Exported unqualified.<br>This means that you can call the function by calling <b>maxx</b> or by calling <b>View.maxx</b> .                                                                                               |
| See also    | <b>Draw.Dot</b> for an example of the use of <b>maxx</b> and for a diagram illustrating x and y positions.                                                                                                               |

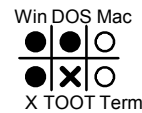
# View.maxy



|             |                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>maxy : int</b>                                                                                                              |
| Description | The <b>maxy</b> function is used to determine the maximum value of y for the current graphics mode.                            |
| Example     | This program outputs the maximum y value.<br><pre>setscreen ("graphics")<br/>...<br/>put "The maximum y value is ", maxy</pre> |

- Details** The screen should be in a "graphics" mode. If it is not, it will automatically be set to "graphics" mode. See **setscreen** for details.  
For the default IBM PC compatible graphics mode (CGA), **maxy** = 199.
- Status** Exported unqualified.  
This means that you can call the function by calling **maxy** or by calling **View.maxy**.
- See also** **Draw.Dot** for an example of the use of **maxy** and for a diagram illustrating x and y positions.

## View.Set



- Syntax** **View.Set ( s : string )**
- Example** Here are example uses of the **View.Set** procedure. In many cases, these will appear as the first statement of the program. However, they can appear any place in a program.
- ```
View.Set ("graphics")      % Switch to graphics mode
View.Set ("graphics:ega")  % Switch to EGA graphics mode
View.Set ("screen")        % Switch to screen mode
View.Set ("nocursor")      % Turn off cursor
View.Set ("noecho")        % Do not echo keystrokes
```
- Description** The **View.Set** statement is used to change the mode of the screen, as well as the way in which Turing does input and output. The parameter to **View.Set** is a string, such as "graphics". The string contains one or more options separated by commas, such as "text, noecho". On a window system (WinOOT, MacOOT, X OOT), **View.Set** affects the active window. On a screen (DOS OOT, TermOOT), it affects the screen.
- Details** Many of the options to **View.Set** are specific to IBM PC compatible computers and Apple Macintoshes. They have been adapted to other systems such as the Icon and X Windows.
There are three screen modes, **text**, **screen** and **graphics**. Most systems do not support all three modes.

System	text	screen	graphics	
WinOOT			X	(Default: 640x300)
DOS OOT		X	X	(Default: screen)
MacOOT			X	(Default: 480x275)
X OOT			X	(Default: 720x375)
TOOT	X			(Default: text)

TermOOT		X		(Default: screen)
---------	--	---	--	-------------------

text mode does not allow any character graphics whatsoever. Only **put** and **get** are allowed.

screen mode allows character graphics commands such as **Text.Locate** and **Text.Color**.

graphics mode allows character graphics and pixel graphics commands such as **Text.Locate** and **Draw.Box**.

If a character graphics command is given while in **text** mode, the program automatically switches to **screen** mode if available, otherwise it switches to **graphics** mode. If a pixel graphics command is given while in **text** or **screen** mode, the program automatically switches to **graphics** mode.

Where the options to **View.Set** are mutually exclusive, they are listed here with the default underlined. Here are the options:

"**cursor**", "**nocursor**" - Causes the cursor to be shown (or hidden).

At the time of writing, there is no visible cursor in graphics mode under DOS OOT or under MacOOT. This will be changed in later versions.

There is an optional suffix for "**cursor**" that determines the shape of the cursor. In DOS OOT, the cursor is constructed out of horizontal lines numbered 0, 1, 2, up to 7, with 0 being the top. The suffix gives the range of lines to be used for the cursor, for example, "**cursor:5;7**" specifies a cursor consisting of lines 5 through 7. In general, this form is "**cursor:startline;endline**", where startline and endline are integer literals such as 5 and 7. On the Apple Macintosh, it is possible to set the cursor size from 0-10.

Under DOS OOT and MacOOT, you can also specify the cursor to be a block or an underline by using "**cursor:block**" or "**cursor:underline**".

"**echo**", "**noecho**" - Causes (or suppresses) echoing of characters that are typed. Echoing is commonly turned off in interactive programs to keep typed characters from being echoed at inappropriate places on the screen.

"**noxor**", "**xor**" - **noxor** mode means that all drawing is done normally. In **xor** mode, all pixel graphics are drawn XOR'ed on the background (with the exception of the **Pic** routines, where the drawing mode is specified). The most important property of an XOR'ed object is that it can be erased and the background restored by XOR'ing the object on top of itself.

"**visible**", "**invisible**", "**popup**" - Causes the active window to become visible (invisible or, for popup, invisible until input or output occurs in the window). This applies only to systems with windows (WinOOT, X OOT and MacOOT).

"**title:<text>**" - Causes the title of the active window to be set to **<text>**. This applies only to systems with windows (WinOOT, X OOT and MacOOT).

"**position:**<*x*>;<*y*>" - Causes the position of the upper left corner of the active window to be set to (*x*, *y*). This applies only to systems with windows (WinOOT, X OOT and MacOOT).

"**text**", "**screen**", "**graphics**" - Sets mode to the given mode and always erases the screen, even when already in the requested mode; "*text*" is the default character output mode; "*screen*" is character graphics mode, allowing the **locate** procedure; "*graphics*" is "*pixel graphics*" mode.

Under WinOOT and X OOT, **screen** mode can have a modifier in the form "*screen:*<*rows*>;<*cols*>". This sets the window to be <*rows*> by <*cols*> in size. At the time of writing, MacOOT did not support this feature. Check your release notes to see if any changes have occurred.

Under WinOOT and X OOT, **graphics** mode can have a modifier in the form "*graphics:*<*x*>;<*y*>". This sets the window to be <*x*> by <*y*> in size. At the time of writing, MacOOT did not support this feature. Check your release notes to see if any changes have occurred.

Under a future release of MacOOT, **graphics** mode can have a modifier in the form "*graphics:*<*x*>;<*y*>;<*depth*>". This sets the window to be <*x*> by <*y*> in size. <*depth*> has the following allowable values:

bw or **1** or **2** = allows 2 colors (black and white)
4 = allows 4 colors
8 or **256** = allows 256 colors
16 = allows 16 colors
thousands = 16 bit color (thousands of colors)
millions = 32 bit color (millions of colors)
cga = The window will use the 4 color CGA palette.
ega or **vga** = The window will use the 16 color VGA palette.
mcga = The window will use the 256 color MCGA palette.

Note that the depths that use IBM color palettes have a window with a black background. They also use the IBM characters (i.e. character sizes are 8x8, 8x14 or 8x16 as they are on the IBM PC).

The set of "*graphics*" options is given on the next page. On Apple Macintoshes, the graphics screen is 480x275 by default. The size of the screen can also be set with a suffix in the form "*graphics:*150;250", which would set the graphics output window to be 150x250 pixels. Macintoshes also allow you to set the window size in *screen* mode. The default is 25 rows by 80 columns but this can be changed with "*screen:*10;100" to be 10 rows by 100 columns.

	<u>Mode</u>		<u>maxx+1</u>	<u>maxy+1</u>	<u>maxcolor+1</u>
"graphics"	320	200	4		
"graphics:cga"	320	200	4		
"graphics:ega"	640	350	16		
"graphics:e16"	640	350	16		
"graphics:vga"	640	480	16		
"graphics:v16"	640	480	16		
"graphics:mcga"	320	200	256		
"graphics:m256"	320	200	256		
"graphics:svga"	640	480	256 (DOS OOT)		
"graphics:svga1"	640	400	256 (DOS OOT)		

"graphics:svga2"	640	480	256 (DOS OOT)
"graphics:svga3"	800	600	16 (DOS OOT)
"graphics:svga4"	800	600	256 (DOS OOT)
"graphics:svga5"	1024	768	16 (DOS OOT)
"graphics:svga6"	1024	768	256 (DOS OOT)

Status Exported qualified.
This means that you can only call the function by calling **View.Set**, not by calling **Set**.

View.SetActivePage

Win DOS Mac

 X TOOT Term

Syntax **View.SetActivePage** (*page* : int)

Description In certain graphics modes under DOS OOT, it is possible to have multiple graphics pages. This means it is possible to draw to one graphics page while another is visible. You can use **View.SetActivePage** to specify which graphics page is being drawn upon and use **View.SetVisiblePage** to specify the graphics page that is to be visible.

By drawing on a different graphics page from the currently visible page, it is possible to avoid the flicker that might occur while the image is being composed, especially if it is composed of several different parts.

The general method is to make the second page the active page, draw the image upon it, then make the second page visible. Then make the first page active and draw upon it while the user is looking at the second graphics page.

View.SetActivePage sets the currently active graphics page. The default page is 1.

Example This program draws a large number of maple leaves. The user will only see the final image rather than seeing each maple leaf individually drawn.

```
setscreen ("graphics:ega")
assert Config.Display (cdMaxNumPages) > 1
View.SetActivePage (2)    % Set the 2nd page to be active
% The user does not see the individual leaves being drawn
const midx : int := maxx div 2
const midy : int := maxy div 2
for decreasing i : midx .. 0 by 10
  Draw.FillMapleLeaf (midx - i, midy - i, midx + i, midy + i,
                     i mod 8 + 8)
end for
View.SetVisiblePage (2)    % 2nd page is now visible
```


Details	<p>You must use Config.Display in order to determine the number of graphics modes available. The <i>cdMaxNumPages</i> parameter will return the number of graphics pages available.</p> <p>At the time of writing, only “ega” graphics mode had more than one graphics page. This may change with future releases. Consult the release notes for any changes that may have occurred.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling View.SetActivePage, not by calling SetActivePage.</p>
See also	View.SetVisiblePage , View.GetActivePage and View.GetVisiblePage for routines to switch active and visible graphics pages.

View.SetVisiblePage



Syntax	View.SetVisiblePage (<i>page</i> : int)
Description	<p>In certain graphics modes under DOS OOT, it is possible to have multiple graphics pages. This means it is possible to draw to one graphics page while another is visible. You can use View.SetActivePage to specify which graphics page is being drawn upon and use View.SetVisiblePage to specify the graphics page that is to be visible.</p> <p>By drawing on a different graphics page from the currently visible page, it is possible to avoid the flicker that might occur while the image is being composed, especially if it is composed of several different parts.</p> <p>The general method is to make the second page the active page, draw the image upon it, then make the second page visible. Then make the first page active and draw upon it while the user is looking at the second graphics page.</p> <p>View.SetVisiblePage sets the currently visible graphics page. The default page is 1.</p>
Example	<p>This program draws a large number of maple leaves. The user will only see the final image rather than seeing each maple leaf individually drawn.</p> <pre> setscreen ("graphics:ega") assert Config.Display (cdMaxNumPages) > 1 View.SetActivePage (2) % Set the 2nd page to be active % The user does not see the individual leaves being drawn const midx : int := maxx div 2 const midy : int := maxy div 2 </pre>

```

for decreasing i : midx .. 0 by 10
  Draw.FillMapleLeaf (midx - i, midy - i, midx + i, midy + i,
                     i mod 8 + 8)
end for

View.SetVisiblePage (2)      % 2nd page is now visible

```

Details	<p>You must use Config.Display in order to determine the number of graphics modes available. The <i>cdMaxNumPages</i> parameter will return the number of graphics pages available.</p> <p>At the time of writing, only “ega” graphics mode had more than one graphics page. This may change with future releases. Consult the release notes for any changes that may have occurred.</p>
Status	<p>Exported qualified.</p> <p>This means that you can only call the function by calling View.SetVisiblePage, not by calling SetVisiblePage.</p>
See also	View.SetActivePage , View.GetActivePage and View.GetVisiblePage for routines to switch active and visible graphics pages.

View.WhatDotColor



Syntax	View.WhatDotColor (<i>x, y</i> : int) : int
Description	The View.WhatDotColor function is used to determine the color number of the specified pixel. The alternate spelling is View.WhatDotColour .
Example	<p>This program draws a line which bounces off the edges of the screen and makes a beep when it finds a pixel that has already been colored.</p>

```

View.Set ( "graphics" )
var x, y : int := 0
var dx, dy : int := 1
loop
  if View.WhatDotColor ( x, y ) not= 0 then
    Music.Sound ( 400, 50 )
  end if
  Draw.Dot ( x, y, 1 )
  x := x + dx
  y := y + dy
  if x = 0 or x = maxx then
    dx := -dx
  end if
  if y = 0 or y = maxy then
    dy := -dy
  end if
end loop

```

end if
end loop

Details	The screen should be in a "graphics" mode. If is not set to "graphics" mode, it will automatically be set to "graphics" mode. See View.Set for details.
Status	Exported qualified. This means that you can only call the function by calling View.WhatDotColor , not by calling WhatDotColor .
See also	Draw.Dot , which is used for setting the color of a pixel. See also maxx and maxy , which are used to determine the number of pixels on the screen. See also Music.Sound , which causes the computer to make a sound.

wait block a process statement

Syntax	A <i>waitStatement</i> is: wait <i>variableReference</i> [, <i>expn</i>]
Description	The wait statement is used in a concurrent program to cause the executing process to be blocked (to go to sleep) until it is awakened by a signal statement. The statement can only be used inside a monitor (a special kind of module that handles concurrency). A wait statement operates on a condition variable (the <i>variableReference</i>), which is essentially a queue of sleeping processes. See condition for an example of a wait statement.
Details	A wait statement for a priority condition must include the optional <i>expn</i> ,. This expression must be a non-negative int value which is used to order processes waiting for the condition, low numbers first. A wait statement for a timeout condition must include the optional <i>expn</i> , which must be a non-negative int value which gives the <i>timeout interval</i> . A process waiting for a timeout condition is implicitly awakened if it waits longer than its timeout interval.
See also	condition and signal . See also monitor and fork . See also empty . See also pause .

wallclock seconds since 1/1/1970 procedure

Syntax	wallclock (var c : int)
Description	The wallclock statement is used to determine the time in seconds since 00:00:00 GMT (Greenwich Mean Time) January 1, 1970.
Example	<p>This program tells you how many seconds since 1970.</p> <pre>var seconds : string wallclock (seconds) put "The number of seconds since 1970 is ", seconds</pre>
Details	<p>Be warned that on some computers such as IBM PC compatibles or Apple Macintoshes, the time may not be set correctly in the operating system; in that case, the wallclock procedure will give incorrect results. Also, on IBM PC compatibles, the call is dependent on having the time zone TZ variable correctly set. On an IBM PC, the default time zone is set to PST (6 hours from GMT).</p> <p>On the Apple Macintosh, the wallclock procedure returns the number of seconds since 00:00:00 local time Jan. 1, 1970.</p>
See also	<p>delay, time, clock, sysclock and date statements.</p> <p>See also predefined unit Time.</p>

whatcol cursor position function

Syntax	whatcol : int
Description	The whatcol function is used to determine the cursor position's column.
Example	<p>This program outputs <i>The current row is 5, the current column is 15.</i></p> <pre>locate (5, 10) put "12345".. put "The current row is", whatrow</pre>

put "The current column is", **whatcol**

- Details The screen should be in a *"screen"* or *"graphics"* mode. **whatcol** functions properly even if the cursor is invisible.
- See also the **whatrow** function, which is used to set the determine the cursor row. See also the **locate**, **maxrow** and **maxcol** procedure.
See also predefined unit **Text**.

whatcolor text color graphics function

- Syntax **whatcolor : int**
- Description The **whatcolor** function is used to determine the current (foreground) color, ie., the color used for characters that are output using **put**. The alternate spelling is **whatcolour**.
- Example This program outputs the currently-active color number. The message is also given in the currently-active color.
- ```
setscreen ("graphics")
...
 put "This writing is in color number ", whatcolor
```
- Details            The screen should be in a *"screen"* or *"graphics"* mode. See **setscreen** for details.
- See also           the **color** procedure, which is used to set the color. See also **colorback** and **whatcolorback**.  
See also predefined unit **Text**.

## whatcolorback   color of background function

- Syntax            **whatcolorback : int**

|             |                                                                                                                                                                                                                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | The <b>whatcolorback</b> function is used to determine the current background color. The alternate spelling is <b>whatcolourback</b> .                                                                                                                                                                   |
| Example     | <p>This program outputs the currently-active background color number. The background color of the message is determined by this number.</p> <pre> <b>setscreen</b> ( "screen" ) ... <b>put</b> "The background of this writing"            <b>put</b> "is in color number ", <b>whatcolorback</b> </pre> |
| Details     | The screen should be in a "screen" or "graphics" mode. Beware that the meaning of background color is different in these two modes. See <b>colorback</b> for details.                                                                                                                                    |
| See also    | <b>color</b> and <b>whatcolor</b> .<br>See also predefined unit <b>Text</b> .                                                                                                                                                                                                                            |

## whatdotcolor    graphics function

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>whatdotcolor</b> ( <i>x</i> , <i>y</i> : <b>int</b> ) : <b>int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Description | The <b>whatdotcolor</b> function is used to determine the color number of the specified pixel. The alternate spelling is <b>whatdotcolour</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Example     | <p>This program draws a line which bounces off the edges of the screen and makes a beep when it finds a pixel that has already been colored.</p> <pre> <b>setscreen</b> ( "graphics" ) <b>var</b> <i>x</i>, <i>y</i> : <b>int</b> := 0 <b>var</b> <i>dx</i>, <i>dy</i> : <b>int</b> := 1 <b>loop</b>   <b>if</b> <b>whatdotcolor</b> ( <i>x</i>, <i>y</i> ) <b>not=</b> 0 <b>then</b>     <b>sound</b> ( 400, 50 )   <b>end if</b>   <b>drawdot</b> ( <i>x</i>, <i>y</i>, 1 )   <i>x</i> := <i>x</i> + <i>dx</i>   <i>y</i> := <i>y</i> + <i>dy</i>   <b>if</b> <i>x</i> = 0 <b>or</b> <i>x</i> = <b>maxx</b> <b>then</b>     <i>dx</i> := -<i>dx</i>   <b>end if</b>   <b>if</b> <i>y</i> = 0 <b>or</b> <i>y</i> = <b>maxy</b> <b>then</b>     <i>dy</i> := -<i>dy</i>   <b>end if</b> <b>end loop</b> </pre> |

|          |                                                                                                                                                                                                                                                                                      |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | The screen should be in a <i>"graphics"</i> mode. If is not set to <i>"graphics"</i> mode, it will automatically be set to <i>"graphics"</i> mode. See <b>setscreen</b> for details.                                                                                                 |
| See also | <b>drawdot</b> , which is used for setting the color of a pixel. See also <b>maxx</b> and <b>maxy</b> , which are used to determine the number of pixels on the screen. See also <b>sound</b> , which causes the computer to make a sound.<br>See also predefined unit <b>View</b> . |

## whatpalette graphics function



|             |                                                                                                                                                                                                   |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>whatpalette : int</b>                                                                                                                                                                          |
| Description | The <b>whatpalette</b> function is used to determine the current palette number.                                                                                                                  |
| Example     | This program outputs the current palette number.<br><pre> <b>setscreen</b> ( "graphics" ) ... <b>put</b> "The current palette number is ", <b>whatpalette</b> </pre>                              |
| Details     | The <b>whatpalette</b> function is meaningful only in a <i>"graphics"</i> mode.                                                                                                                   |
| See also    | the <b>setscreen</b> procedure for a description of the graphics modes. See also the <b>palette</b> statement, which is used to set the palette number.<br>See also predefined unit <b>View</b> . |

## whatrow cursor position function

|             |                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>whatrow : int</b>                                                                                                                     |
| Description | The <b>whatrow</b> function is used to determine the cursor position's row.                                                              |
| Example     | This program outputs <i>The current row is 5, the current column is 15.</i><br><pre> <b>locate</b> ( 5, 10 ) <b>put</b> "12345".. </pre> |

```

put "The current row is", whatrow
 put "The current column is", whatcol

```

|          |                                                                                                                                                                                              |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | The screen should be in a "screen" or "graphics" mode. <b>whatrow</b> functions properly even if the cursor is invisible.                                                                    |
| See also | the <b>whatcol</b> function, which is used to determine the cursor column. See also the <b>locate</b> , <b>maxrow</b> and <b>maxcol</b> procedure.<br>See also predefined unit <b>Text</b> . |

## whattextchar graphics function

Char graphics only

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>whattextchar : string (1)</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Description | The <b>whattextchar</b> function is used to determine the character on the screen at the current location.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Example     | <p>This program outputs a message and then changes the foreground color (the color of the letters) of the message to color number 1 and the background color (surrounding each letter) to color number 7. The actual message (each letter) is not changed.</p> <pre> <b>setscreen</b> ( "screen" ) <b>const</b> message := "Happy New Year!!" <b>put</b> message ... <b>for</b> column : 1 .. <b>length</b> ( message )   <b>locate</b> ( 1, column )   <b>color</b> ( 1 )           % Color of letter   <b>colorback</b> ( 7 )       % Color around letter   <b>put</b> <b>whattextchar</b> ..   % Use same letter <b>end for</b> </pre> |
| Details     | The <b>whattextchar</b> function is meaningful only in "screen" mode. In "graphics" mode, the concept of <i>text</i> on the screen is replaced by the concept of <i>pixels</i> on the screen.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| See also    | <b>setscreen</b> which describes modes. See also <b>color</b> , <b>whattextcolor</b> , and <b>whattextcolorback</b> .<br>See also predefined unit <b>Text</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |



## whattextcolor graphics function

Char graphics only

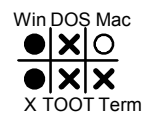
|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>whattextcolor : int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Description | The <b>whattextcolor</b> function is used to determine the color of the character on the screen at the current location. The alternate spelling is <b>whattextcolour</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Example     | <p>This program prints out a message with each letter in a random color, and then prints the same message on the next line in exactly the reverse colors.</p> <pre>setscreen ( "screen" ) var clr : int const message := "Happy New Year!!" for column : 1 .. length ( message )     randint ( clr, 1, maxcolor )     color ( clr )     locate ( 1, column )     put message ( i ) .. end for locate ( 1, 1 ) for column : 1 .. length ( message )     locate ( 1, length ( message ) + 1 - c )     clr := whattextcolor     color ( clr )     locate ( 2, column )     put message ( i ) .. end for</pre> |
| Details     | The <b>whattextcolor</b> function is meaningful only in "screen" mode. In "graphics" mode, the concept of <i>text</i> on the screen is replaced by the concept of <i>pixels</i> on the screen.                                                                                                                                                                                                                                                                                                                                                                                                             |
| See also    | <b>setscreen</b> which describes modes. See also <b>color</b> , <b>whattextchar</b> , and <b>whattextcolorback</b> .<br>See also predefined unit <b>Text</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                             |

## whattextcolorback graphics function

Char graphics only

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>whattextcolorback : int</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Description | The <b>whattextcolorback</b> function is used to determine the background color of the character on the screen at the current location. The alternate spelling is <b>whattextcolourback</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Example     | <p>This program prints out a message with each letter in a random background color, and then prints the same message on the next line in exactly the reverse background colors.</p> <pre> <b>setscreen</b> ( "screen" ) <b>var</b> <i>clr</i> : <b>int</b> <b>const</b> <i>message</i> := "Happy New Year!!" <b>for</b> <i>column</i> : 1 .. <b>length</b> ( <i>message</i> )     <b>randint</b> ( <i>clr</i>, 1, <b>maxcolor</b> )     <b>colorback</b> ( <i>clr</i> )     <b>locate</b> ( 1, <i>column</i> )     <b>put</b> <i>message</i> ( <i>i</i> ) .. <b>end for</b> <b>locate</b> ( 1, 1 ) <b>for</b> <i>column</i> : 1 .. <b>length</b> ( <i>message</i> )     <b>locate</b> ( 1, <b>length</b> ( <i>message</i> ) + 1 - <i>c</i> )     <i>clr</i> := <b>whattextcolorback</b>     <b>colorback</b> ( <i>clr</i> )     <b>locate</b> ( 2, <i>column</i> )     <b>put</b> <i>message</i> ( <i>i</i> ) .. <b>end for</b></pre> |
| Details     | The <b>whattextcolorback</b> function is meaningful only in "screen" mode. In "graphics" mode, the concept of <i>text</i> on the screen is replaced by the concept of <i>pixels</i> on the screen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| See also    | <b>setscreen</b> which describes modes. See also <b>color</b> , <b>whattextchar</b> , and <b>whattextcolor</b> .<br>See also predefined unit <b>Text</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

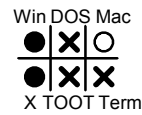
## Window



|              |                                                                                                                                                                                                                                                                     |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description  | <p>This unit contains the predefined subprograms that handle windows. There are routines to open, close, hide, show and select windows.</p> <p>All routines in the <b>Window</b> unit are exported qualified (and thus must be prefaced with "<b>Window.</b>").</p> |
| Entry Points | <p><b>Open</b>                      Opens a new execution window.</p>                                                                                                                                                                                               |

|                    |                                                             |
|--------------------|-------------------------------------------------------------|
| <b>Close</b>       | Closes an execution window.                                 |
| <b>Select</b>      | Selects an execution window for output.                     |
| <b>GetSelect</b>   | Returns the currently-selected execution window.            |
| <b>SetActive</b>   | Selects and activate (make front-most) an execution window. |
| <b>GetActive</b>   | Gets the current active window.                             |
| <b>GetPosition</b> | Get the screen position of an execution window.             |
| <b>SetPosition</b> | Set the screen position of an execution window.             |
| <b>Hide</b>        | Hides an execution window.                                  |
| <b>Show</b>        | Shows the current execution window.                         |
| <b>Set</b>         | Sets the configuration of the execution window.             |

## Window.Close



|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Window.Close</b> ( <i>windowID</i> : int)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Description | The <b>Window.Close</b> procedure closes the window specified by the <i>windowID</i> parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Example     | <p>The following program opens a window, makes it active and then closes the window after getting a keystroke from the user.</p> <pre> % Open the window var winID : int winID := Window.Open ("position:300;300,graphics:200;200")  % Draw the random ovals in the box var x, y, clr : int for : 1 .. 20     x := Rand.Int (0, maxx)      % Random x     y := Rand.Int (0, maxy)      % Random y     clr := Rand.Int (0, maxcolor) % Random color     Draw.FillOval (x, y, 30, 30, clr) end for  var ch : char := getchar          % Wait for input </pre> |

**Window.Close** (*winID*)                      % Close the window

|          |                                                                                                                                          |
|----------|------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | If a window is selected (i.e. output is going to that window) when it is closed, the main <b>Run</b> window becomes the selected window. |
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Window.Close</b> , not by calling <b>Close</b> .     |
| See also | <b>Window.Open</b> and <b>Window.Select</b> .                                                                                            |

## Window.GetActive

Win DOS Mac  
● × ○  
○ × ×  
X TOOT Term

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Window.GetActive</b> : int                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Description | The <b>Window.GetActive</b> function returns the window ID of the active window. If the active window is not a run window, then it returns -1 and sets <b>Error.Last</b> and <b>Error.LastMsg</b> to indicate the fact.<br><br>An active window is defined as the window that has the input focus. This means that any typing will be sent to the active window. Under most systems an active window is indicated by a change in the appearance of the window. |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Window.GetActive</b> , not by calling <b>GetActive</b> .                                                                                                                                                                                                                                                                                                                   |
| See also    | <b>Window.SetActive</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                      |

## Window.GetPosition

Win DOS Mac  
● × ○  
○ × ×  
X TOOT Term

|        |                                                                                   |
|--------|-----------------------------------------------------------------------------------|
| Syntax | <b>Window.GetPosition</b> ( <i>windowID</i> : int, var <i>x</i> , <i>y</i> : int) |
|--------|-----------------------------------------------------------------------------------|

**Description**      The **Window.GetPosition** procedure returns the location of the specified execution window on the screen in the *x* and *y* parameters. The *x* and *y* parameters specify the lower left corner of the window in screen coordinates. (0, 0) is the lower left corner of the screen.

**Example**            The following program outputs the current position of the run window.

```
% Constants for windows
const titleBarHeight : int := 21
const windowEdgeSize : int := 13

% Calculate the actual size of a window
var windowWidth : int := maxx + windowEdgeSize
var windowHeight : int := maxy + windowEdgeSize + titleBarHeight

% Get the screen size
var screenWidth : int := Config.Display (cdScreenWidth)
var screenHeight : int := Config.Display (cdScreenHeight)

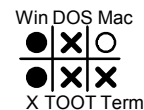
% Open the window
var winID : int := Window.Open ("title:Upper Right")
Window.SetPosition (winID, screenWidth - windowWidth, screenHeight -
windowHeight)

% Return the current position
var windowXPosition, windowYPosition : int
Window.GetPosition (winID, windowXPosition, windowYPosition)
put "Window located at ", windowXPosition, ", ", windowYPosition
```

**Status**              Exported qualified.  
This means that you can only call the function by calling **Window.GetPosition**, not by calling **GetPosition**.

**See also**            **Window.SetPosition** to set the current window position and **Config.Display** to get the size of the screen.

## Window.GetSelect



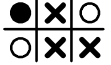
**Syntax**            **Window.GetSelect** : **int**

**Description**        The **Window.GetSelect** function returns the window ID of the selected window. If the select window is the main run window, then it returns *winDefaultID*.

A selected window is defined as the window that output will be sent to. It can be invisible. When a program starts execution, the selected window is the main **Run** window.

|          |                                                                                                                                              |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Status   | Exported qualified.<br>This means that you can only call the function by calling <b>Window.GetSelect</b> , not by calling <b>GetSelect</b> . |
| See also | <b>Window.Select</b> .                                                                                                                       |

## Window.Hide

Win DOS Mac  
  
 X TOOT Term

|             |                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Window.Hide</b> ( <i>windowID</i> : <b>int</b> )                                                                                                                                                                                                                                                                                                                   |
| Description | The <b>Window.Hide</b> procedure hides the specified window. This means it disappears from the user's screen. However, it is still possible to select and draw the window while it remains hidden. If the user activates it (using <b>Window.GetActive</b> ) it will automatically appear.<br>To make a window appear after it's hidden, you use <b>Window.Show</b> . |
| Details     | When a window is hidden, output to it is faster. It is quite possible for the you to hide a window, do complicated drawing to it and then make it appear in order to have the program execute faster.                                                                                                                                                                 |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Window.Hide</b> , not by calling <b>Hide</b> .                                                                                                                                                                                                                                    |
| See also    | <b>Window.Select</b> and <b>Window.SetActive</b> .                                                                                                                                                                                                                                                                                                                    |

## Window.Open

Win DOS Mac  
  
 X TOOT Term

|        |                                                                        |
|--------|------------------------------------------------------------------------|
| Syntax | <b>Window.Open</b> ( <i>setUpString</i> : <b>string</b> ) : <b>int</b> |
|--------|------------------------------------------------------------------------|

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | <p>The <b>Window.Open</b> function is used to create a window. A window ID is returned if the window is successfully created. If the window is not created then it returns 0. <b>Error.Last</b> and <b>Error.LastMsg</b> can then be used to determine the cause of the failure.</p> <p>The <i>setUpString</i> parameter is identical to that of <b>Window.Set</b>. See <b>Window.Set</b> for the list of options available.</p> <p>When the window is created, it is automatically selected (i.e. all output will be sent to that window unless redirected by a <b>Window.Select</b> command).</p>              |
| Example     | <p>The following program opens a window, makes it active and then close the window after getting a keystroke from the user.</p> <pre> % Open the window var winID : int winID := Window.Open ("position:300;300,graphics:200;200")  % Draw the random ovals in the box var x, y, clr : int for : 1 .. 20     x := Rand.Int (0, maxx)      % Random x     y := Rand.Int (0, maxy)      % Random y     clr := Rand.Int (0, maxcolor) % Random color     Draw.FillOval (x, y, 30, 30, clr) end for  var ch : char := getchar          % Wait for input  Window.Close (winID)              % Close the window </pre> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Window.Open</b>, not by calling <b>Open</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| See also    | <b>Window.Set</b> for the syntax of <i>startUpString</i> . See also <b>Window.Select</b> and <b>Window.SetActive</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## Window.Select

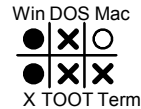


|             |                                                                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Window.Select</b> ( <i>windowID</i> : int)                                                                                                                                                                                                                            |
| Description | <p>The <b>Window.Select</b> selects the window that output is to be sent to.</p> <p>A selected window is defined as the window that output will be sent to. It can be invisible. When a program starts execution, the selected window is the main <b>Run</b> window.</p> |

Status Exported qualified.  
This means that you can only call the function by calling **Window.Select**, not by calling **Select**.

See also **Window.Select** and **Window.SetActive**.

## Window.Set



Syntax **Window.Set** (*windowID* : **int**, *setUpString* : **string**)

Description The **Window.Set** procedure sets the configuration of the window specified by the *windowID* parameter. The *setUpString* parameter can be any combination of the following, separated by commas.

**screen:***<rows>;<cols>* - Changes the screen size to be *<rows>* rows by *<cols>* columns in size.

**graphics:***<xsize>;<ysize>* - Changes the screen size to be *<xsize>* pixels across and *<ysize>* pixels in height.

**graphics:***<mode>* - Changes the screen to mimic an IBM graphics mode in size and color palette.

| <u>Mode</u> |     | <u>maxx+1</u> | <u>maxy+1</u> | <u>maxcolor+1</u> |
|-------------|-----|---------------|---------------|-------------------|
| "cga"       | 320 | 200           | 4             | (CGA)             |
| "mono"      | 320 | 200           | 4             | (gray)            |
| "hmono"     | 640 | 200           | 2             |                   |
| "16"        | 320 | 200           | 16            |                   |
| "h16"       | 640 | 200           | 16            |                   |
| "ega"       | 640 | 350           | 16            |                   |
| "v2"        | 640 | 480           | 2             |                   |
| "vga"       | 640 | 480           | 16            |                   |
| "mcga"      | 320 | 200           | 256           |                   |

**visible** | **invisible** | **popup** - Sets the screen to be visible, invisible or popup. A popup window is hidden until output is sent to that window. The main **Run** window is a popup window. If you never send any output to it, it never appears.

**noxor** | **xor** - Sets whether all drawing operations draw using XOR.

**nocursor** | **cursor** - Sets whether the cursor is visible or not.

**noecho** | **echo** - Sets whether the input from the keyboard is echoed to the screen.



**title:***<text>* - Sets the window title bar to *<text>*.

**position:***<x>;<y>* - Sets the position of the top left corner of the window to be (*<x>*, *<y>*).

Status           Exported qualified.

This means that you can only call the function by calling **Window.Set**, not by calling **Set**.

See also       **Window.Open** and **View.Set**.

## Window.SetActive



Syntax       **Window.SetActive** (*windowID* : **int**)

Description   The **Window.SetActive** procedure activates the window specified by the *windowID* parameter.

An active window is defined as the window that has the input focus. This means that any typing will be sent to the active window. Under most systems an active window is indicated by a change in the appearance of the window.

Details       In general, it is unwise to change the active window. If the user is working on another program at the same time the program is running and the program executes the **Window.SetActive** procedure, she or he will suddenly be returned to OOT without warning.

Status       Exported qualified.

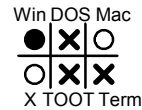
This means that you can only call the function by calling **Window.SetActive**, not by calling **SetActive**.

See also       **Window.GetActive** and **Window.Select**.

# Window.SetPosition

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Window.SetPosition</b> ( <i>windowID</i> : int, <i>x</i> , <i>y</i> : int)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Description | The <b>Window.SetPosition</b> procedure moves the location of the specified execution window on the screen. <i>x</i> and <i>y</i> specify the lower left corner of the window in screen coordinates. (0, 0) is the lower left corner of the screen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Example     | <p>The following program opens four windows, one at each corner of the screen.</p> <pre> % Constants for windows const titleBarHeight : int := 21 const windowEdgeSize : int := 13  % Calculate the actual size of a window var windowWidth : int := maxx + windowEdgeSize var windowHeight : int := maxy + windowEdgeSize + titleBarHeight  % Get the screen size var screenWidth : int := Config.Display (cdScreenWidth) var screenHeight : int := Config.Display (cdScreenHeight)  % Open the window var winID1 : int := Window.Open ("title:Upper Right") Window.SetPosition (winID1, screenWidth - windowWidth,   screenHeight - windowHeight)  var winID2 : int := Window.Open ("title:Upper Left") Window.SetPosition (winID2, 0, screenHeight - windowHeight)  var winID3 : int := Window.Open ("title:Lower Left") Window.SetPosition (winID3, 0, 0)  var winID4 : int := Window.Open ("title:Lower Right") Window.SetPosition (winID4, screenWidth - windowWidth, 0) </pre> |
| Status      | <p>Exported qualified.</p> <p>This means that you can only call the function by calling <b>Window.SetPosition</b>, not by calling <b>SetPosition</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| See also    | <b>Window.GetPosition</b> to get the current window position and <b>Config.Display</b> to get the size of the screen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

# Window.Show



|             |                                                                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <b>Window.Show</b> ( <i>windowID</i> : <b>int</b> )                                                                                                                                                   |
| Description | The <b>Window.Show</b> procedure makes the specified window appear if it was invisible.<br>To make a window disappear after it's visible, you use <b>Window.Hide</b> .                                |
| Details     | When a window is hidden, output to it is faster. It is quite possible for the you to hide a window, do complicated drawing to it and then make it appear in order to have the program execute faster. |
| Status      | Exported qualified.<br>This means that you can only call the function by calling <b>Window.Show</b> , not by calling <b>Show</b> .                                                                    |
| See also    | <b>Window.Select</b> and <b>Window.SetActive</b> .                                                                                                                                                    |

## write file statement

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | A <i>writeStatement</i> is:<br><b>write</b> : <i>fileNumber</i> [: <i>status</i> ], <i>writeItem</i> {, <i>writeItem</i> }                                                                                                                                                                                                                                                                                                                                                            |
| Description | The <b>write</b> statement outputs each of the <i>writeItems</i> to the specified file. These items are output directly using the <i>binary</i> format that they have in the computer. In other words, the items are not in source (ASCII or EBCDIC) format. In the common case, these items will later be input from the file using the <b>read</b> statement. By contrast, the <b>get</b> and <b>put</b> statements use source format, which a person can read using a text editor. |
| Example     | This example shows how to output a complete employee record using a <b>write</b> statement.<br><pre>var employeeRecord :<br/>  record<br/>    name : string ( 30 )</pre>                                                                                                                                                                                                                                                                                                              |

```

 pay : int
 dept : 0 .. 9
 end record
var fileNo : int
open : fileNo, "payroll", write
...
 write : fileNo, employeeRecord

```

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Details  | <p>An array, record or union may be read and written as a whole. The <i>fileNumber</i> must specify a file that is open with <b>write</b> capability (or else a program argument file that is implicitly opened).</p> <p>The optional <i>status</i> is an <b>int</b> variable that is set to implementation-dependent information about the <b>write</b>. If <i>status</i> is returned as zero, the <b>write</b> was successful. If <i>status</i> is not returned as zero, <i>status</i> gives information about the incomplete or failed <b>write</b> (which is not documented here). Programmers often use <i>status</i> when they are writing a record or array to a file and are not sure if there is enough room on the disk to hold the item. If there is not enough room, the <b>write</b> will fail part way through, but the program can continue and diagnose the problem by inspecting <i>status</i>.</p> <p>A <i>writeItem</i> is:</p> <pre>reference [ : requestedSize [ : actualSize ] ]</pre> <p>Each <i>writeItem</i> is a variable or constant, to be written in internal form. The optional <i>requestedSize</i> is an integer expression giving the number of bytes of data to be written. The <i>requestedSize</i> should be less than or equal to the size of the item's internal form in memory (if it is not, a warning message is issued). If no <i>requestedSize</i> is given, the size of the item in memory is used. The optional <i>actualSize</i> is set to the number of bytes actually written.</p> |
| See also | <b>write</b> , <b>open</b> , <b>close</b> , <b>seek</b> , <b>tell</b> , <b>get</b> and <b>put</b> statements.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

## XOR exclusive "or" operator

|             |                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Syntax      | <i>A</i> <b>xor</b> <i>B</i>                                                                                                                                                                                                                                                                                                                                                                |
| Description | <p>When applied to set values, <b>xor</b> (symmetric difference) yields a set which includes element <i>e</i> if and only if <i>e</i> is contained in exactly one of the operands. When applied to non-negative integer values, <b>xor</b> yields a natural number whose bits are the xor of the corresponding bits of the operands. Both operands <i>A</i> and <i>B</i> are evaluated.</p> |

Example            Status *s3* will contain elements that are in *s1* or *s2* but not both. Here **xor** is a set operator. See **enum** and **set** types for explanations of these types.

```
type status : enum (ready, sending, repeating)
type statusSet : set of status
var s1, s2, s3 : statusSet
s1 := statusSet (status.read, status.sending)
s2 := statusSet (status.read, status.repeating)
...
s3 := s1 xor s2 % Same as (s1 + s2) - (s1 * s2)
```

Example            Each bit of natural number *n3* will be 1 if exactly one of the corresponding bits of *n1* and *n2* are 1. For example, if *n1* = 2#110 (6) and *n2* = 2#010 (2), *n3* will be set to 2#100 (4). Here **xor** is an integer operator.

```
var n1, n2, n3 : nat
...
n3 := n1 xor n2
```

Details            The **xor** operator is not a short circuit operator; in other words, both of its operands are always evaluated. The precedence of **xor** is the same as that of plus (+).

See also            **set**. See also *explicitIntegerConstant* which describes values such as 2#110.



# Appendix A

## Predefined Functions and Procedures

|                    |               |                   |               |                |
|--------------------|---------------|-------------------|---------------|----------------|
| abs                | addr          | arctan            | arctand       | anyclass       |
| break              | buttonchoose  | buttonmoved       | buttonwait    | ceil           |
| chr                | clock         | cls               | color         | colorback      |
| colour             | colourback    | cos               | cosd          | date           |
| delay              | drawarc       | drawbox           | drawdot       | drawfill       |
| drawfillarc        | drawfillbox   | drawfillmapleleaf |               | drawfilloval   |
| drawfillpolygon    | drawfillstar  | drawline          | drawmapleleaf | drawoval       |
| drawpic            | drawpolygon   | drawstar          | empty         | eof            |
| erealstr           | exp           | fetcharg          | floor         | frealstr       |
| getch              | getchar       | getenv            | getpid        | getpriority    |
| hasch              | index         | intreal           | intstr        | length         |
| ln                 | locate        | locatexy          | lower         | max            |
| maxcol             | maxcolor      | maxcolour         | maxint        | maxnat         |
| maxrow             | maxx          | maxy              | min           | minint         |
| minnat             | mousehide     | mousetshow        | mousewhere    | nargs natreal  |
| natstr             | nil           | ord               | palette       | play playdone  |
| pred               | rand          | randint           | randnext      | randomize      |
| randseed           | realstr       | repeat            | round         | setpriority    |
| setscreen          | sign          | simutime          | sin           | sind sizeof    |
| sizepic            | sound         | sqr               | strint        | strintok       |
| strnat             | strnatok      | strreal           | strrealok     | succ sysclock  |
| sysexit            | system        | takepic           | time          | upper          |
| wallclock          | whatcol       | whatcolor         | whatcolorback | whatcolour     |
| whatcolourback     | whatdotcolor  | whatdotcolour     | whatpalette   | whatrow        |
| whattextchar       | whattextcolor | whattextcolorback |               | whattextcolour |
| whattextcolourback |               |                   |               |                |

## Predefined OOT Units

|         |             |          |          |             |
|---------|-------------|----------|----------|-------------|
| Brush   | Button      | CheckBox | Comm     | Concurrency |
| Config  | Dir         | Draw     | DropBox  | EditBox     |
| Error   | ErrorNum    | Event    | File     | Font        |
| GUI     | Input       | Joytick  | Keyboard | Limits      |
| ListBox | Math        | Menu     | Mouse    | Music       |
| Net     | Obsolete    | PC       | Pen      | Pic         |
| Print   | RadioButton | Rand     | RGB      | Sound       |
| Sprite  | Str         | Stream   | Student  | Sys         |
| Text    | Time        | TypeConv | Video    | View Window |

## Predefined OOT Constants

(... means several constants with the prefix, see the module for a complete list)

|                       |                    |                  |                       |
|-----------------------|--------------------|------------------|-----------------------|
| black                 | blue               | brightblue       | brightcyan            |
| brightgreen           | brightmagenta      | brightpurple     | brightred             |
| brightwhite           | brown              | brushErrorBase   | cdMaxNumColors        |
| cdMaxNumColours       | cdMaxNumPages      | cdScreenHeight   | cdScreenWidth         |
| clLanguageVersion     | clMaxNumDirStreams |                  |                       |
| clMaxNumRunTimeArgs   |                    | clMaxNumStreams  | clRelease             |
| cmFPU                 | cmOS               | cmProcessor      | colorbg               |
| colourbg              | colorfg            | colourfg         | configErrorBase       |
| cyan                  | darkgray           | darkgrey         | defWinID              |
| dirErrorBase          | drawErrorBase      | e... (ErrorNum)  | errWinID              |
| excp... (Exceptions)  | fileErrorBase      | fontDefaultID    | fontErrorBase         |
| fontVGAID             | fsysErrorBase      | generalErrorBase | gray            green |
| grey                  | guiErrorBase       | joystick1        | joystick2             |
| lexErrorBase          | magenta            | mouseErrorBase   | musicErrorBase        |
| ootAttr... (File)     | ootk... (Keyboard) | penErrorBase     | picCopy               |
| picErrorBase          | picMerge           | picUnderMerge    | picXor                |
| placeCenterDisplay    | placeCentreWindow  |                  | printerErrorBase      |
| purple                | red                | rgbErrorBase     | spriteErrorBase       |
| streamErrorBase       | textErrorBase      | timeErrorBase    |                       |
| unixSignalToException |                    | viewErrorBase    | white                 |
| windowErrorBase       | yellow             |                  |                       |



## Keywords

|             |             |          |            |             |
|-------------|-------------|----------|------------|-------------|
| addressint  | all         | and      | array      | asm         |
| assert      | begin       | bind     | bits       | body        |
| boolean     | break       | by       | case       | char cheat  |
| checked     | class       | close    | collection | condition   |
| const       | decreasing  | def      | deferred   | div else    |
| elseif      | elsif       | end      | endfor     | endif       |
| endloop     | enum        | exit     | export     | external    |
| false       | fcn         | flexible | for        | fork        |
| forward     | free        | function | get        | handler     |
| if          | implement   | import   | in         | include     |
| inherit     | init        | int      | int1       | int2        |
| int4        | invariant   | label    | loop       | mod module  |
| monitor     | nat         | nat1     | nat2       | nat4 new    |
| not         | objectclass | of       | opaque     | open        |
| or          | packed      | pause    | pervasive  | pointer     |
| post        | pre         |          | priority   | proc        |
| procedure   | process     | put      |            | quit read   |
| real        | real4       | real8    | record     | register    |
| rem         | result      | return   | seek       | self set    |
| shl         | shr         |          | signal     | skip string |
| tag         | tell        |          | then       | timeout     |
| to          | true        | type     | unchecked  | union       |
| unqualified | var         | wait     | when       | write       |
| xor         |             |          |            |             |

## Appendix B

### Selected OOT Predefined Subprograms by Module

|                    |                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Concurrency</b> | Contains the predefined subprograms that deal with concurrency.                                                                                           |
| <b>Config</b>      | Contains the predefined subprograms that deal with getting configuration information about the machine and environment on which the program is being run. |
| <b>Dir</b>         | Contains the predefined subprograms that deal with directories on the operating system level (i.e. creating, deleting and listing directories, etc.).     |
| <b>Draw</b>        | Contains the predefined subprograms that deal with drawing pixel graphics to the screen.                                                                  |
| <b>Error</b>       | Contains the predefined subprograms that deal with errors returned from predefined subprograms.                                                           |
| <b>File</b>        | Contains the predefined subprograms that deal with manipulating files on the operating system level (i.e. copying, deleting and renaming files, etc.).    |
| <b>Font</b>        | Contains the predefined subprograms that deal with drawing text in different fonts, sizes and styles.                                                     |
| <b>GUI</b>         | Contains the predefined subprograms that deal with creating and manipulating a graphical user interface.                                                  |
| <b>Input</b>       | Contains the predefined subprograms that deal with getting input from the keyboard.                                                                       |
| <b>Joystick</b>    | Contains the predefined subprograms that deal with reading the location and button status of the joystick.                                                |
| <b>Limits</b>      | Contains the predefined constants and subprograms that deal with numerical limits and the mathematical accuracy of OOT.                                   |
| <b>Math</b>        | Contains the predefined subprograms that deal with mathematics.                                                                                           |
| <b>Mouse</b>       | Contains the predefined subprograms that deal with using the mouse in an OOT program.                                                                     |
| <b>Music</b>       | Contains the predefined subprograms that deal with sound and music.                                                                                       |
| <b>Net</b>         | Contains the predefined subprograms that deal with connecting and communicating between two TCP/IP networked machines.[                                   |
| <b>PC</b>          | Contains the predefined subprograms that deal with direct access to the hardware under the IBM PC architecture (available only under DOS OOT).            |

|                 |                                                                                                                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Pic</b>      | Contains the predefined subprograms that deal with taking pictures of part of the screen, displaying them and moving pictures from file to screen and back.                                    |
| <b>Rand</b>     | Contains the predefined subprograms that deal with random numbers.                                                                                                                             |
| <b>RGB</b>      | Contains the predefined constants and subprograms that deal with the basic colors and changing the color palette.                                                                              |
| <b>Sprite</b>   | Contains the predefined subprograms that deal with creating, moving and changing sprites (self-contained graphical objects that move over or under the background).                            |
| <b>Str</b>      | Contains the predefined subprograms that deal with manipulating strings.                                                                                                                       |
| <b>Stream</b>   | Contains the predefined subprograms that deal with I/O streams.                                                                                                                                |
| <b>Sys</b>      | Contains the predefined subprograms that deal with the operating system directly (i.e. getting a process id, getting run time arguments and executing commands in the operating system, etc.). |
| <b>Text</b>     | Contains the predefined subprograms that handle character (text) output on the screen (i.e. output using <b>put.</b> ).                                                                        |
| <b>Time</b>     | Contains the predefined subprograms that deal with time (i.e. getting or converting time between formats).                                                                                     |
| <b>TypeConv</b> | Contains the predefined subprograms that deal with converting between standard types.                                                                                                          |
| <b>View</b>     | Contains the predefined subprograms that deal with current output surface. This can be a screen (as in DOS OOT) or a Window (as in X Windows, MS Windows and Macintosh OOT).                   |
| <b>Window</b>   | Contains the predefined subprograms that deal with Windows (i.e. opening, closing, hiding and showing windows, etc.).                                                                          |

## Concurrency

**empty** (*variableReference* : **condition**) :  
**boolean**

Return true if no processes are waiting on the condition queue.

**getpriority** : **nat**

Return the priority of the current process.

**setpriority** (*p* : **nat**)

Set the priority of the current process.

**simutime** : **int**

Return the number of simulated time units that have passed.

## Config

**Config.Display** (*displayCode* : **int**) : **int**

Return information about display attached to computer.

## **Config.Lang** (*langCode* : **int**) : **int**

Return information about the language and implementation limitations.

## **Config.Machine** (*machineCode* : **int**) : **int**

Return information about the computer on which the program is running.

## **Dir**

### Getting Directory Listings

## **Dir.Close** (*stream* : **int**)

Close the directory stream.

## **Dir.Get** (*stream* : int) : string

Return the next file name in the directory listing.

## **Dir.GetLong** (*stream* : int, var *entryName* : string, var *size*, *attribute*, *fileTime* : int)

Get the next file name in the directory listing along with the file size, attributes, and the last modification time of the file.

## **Dir.Open** (*pathName* : string) : int

Open a directory stream in order to get a listing of the directory contents.

## Disk Directory Manipulation

## **Dir.Create** (*pathName* : string)

Create a directory.

## **Dir.Delete** (*pathName* : **string**)

Delete a directory.

**Dir** (cont...)

## Execution Directory Manipulation

## **Dir.Change** (*pathName* : **string**)

Change the current execution directory.

## **Dir.Current** : **string**

Return the current execution directory.

**Draw**

## **Draw.Arc** (*x, y, xRadius, yRadius, initialAngle, finalAngle, clr* : **int**)

Draw an arc on screen centred at (*x,y*).

## **Draw.Box** (*x1, y1, x2, y2, clr : int*)

Draw a box on screen.

## **Draw.Cls**

Clear the screen.

## **Draw.Dot** (*x, y, clr : int*)

Draw a dot on screen at (*x,y*).

## **Draw.Fill** (*x, y, fillColor, borderColor : int*)

Fill in a figure of color *borderColor*.

## **Draw.FillArc** (*x, y, xRadius, yRadius, initialAngle, finalAngle, clr : int*)

Draw a filled pie-shaped wedge on screen centered at (*x,y*).



## **Draw.FillBox** (*x1, y1, x2, y2, clr : int*)

Draw a filled box on the screen.

## **Draw.FillMapleLeaf** (*x1, y1, x2, y2, clr : int*)

Draw a filled maple leaf on the screen.

## **Draw.FillOval** (*x, y, xRadius, yRadius, clr : int*)

Draw a filled oval on screen centered at (*x,y*).

## **Draw.FillPolygon** (*x, y : array 1 .. \* of int, n : int, clr : int*)

Draw a filled polygon on the screen.

## **Draw.FillStar** (*x1, y1, x2, y2, clr : int*)

Draw a filled star on the screen.

**Draw.Line** (*x1, y1, x2, y2, clr : int*)

Draw a line on the screen.

**Draw.MapleLeaf** (*x1, y1, x2, y2, clr : int*)

Draw a maple leaf on screen.

**Draw.Oval** (*x, y, xRadius, yRadius, clr : int*)

Draw an oval on screen centered at (*x,y*).

**Draw.Polygon** (*x, y : array 1 .. \* of int, n : int, clr : int*)

Draw a polygon on the screen.

**Draw.Star** (*x1, y1, x2, y2, clr : int*)

Draw a star on the screen.

## Error

### **Error.Last : int**

Return the (integer) error code from the last subprogram call.

### **Error.LastMsg : string**

Return the error message from the last subprogram call.

### **Error.LastStr : string**

Return the constant name for the error code from the last subprogram call.

### **Error.Msg (*errorCode* : int) : string**

Return the error message corresponding to the error code.

### **Error.Str (*errorCode* : int) : string**

Return the constant name corresponding to the error code.

## **Error.Trip** (*errorCode* : **int**)

Abort execution with the specified error code.

### **File**

## **File.Copy** (*srcPathName*, *destPathName* : **string**)

Copy a file to another location.

## **File.Delete** (*pathName* : **string**)

Delete a file.

## **File.DiskFree** (*pathName* : **string**) : **int**

Return the free disk space in which the file or directory resides.

## **File.Exists** (*pathName* : **string**) : **boolean**

Return true if the file exists.

**File.Rename** (*srcPathName*, *destName* : **string**)

Rename a file or directory

**File.Status** (*pathName* : **string**, **var** *size*,  
*attribute*, *fileTime* : **int**)

Get the size, attributes, and the last modification time of the file.

**Font**

**Font.Draw** (*textStr* : **string**, *x*, *y*, *fontID*,  
*clr* : **int**)

Draw text at position (*x,y*) with a font.

**Font.Free** (*fontID* : **int**)

Free a font.

**Font.Name** (*fontID* : **int**) : **string**

Return the name of the font associated with *fontID*.

**Font.New** (*fontSelectStr* : string) : int

Return the fontID for a specified font string.

**Font.Sizes** (*fontID* : int, var *height*,  
*ascent*, *descent*, *internalLeading* : int)

Return information about a font.

**Font** (cont...)

**Font.Width** (*textStr* : string, *fontID* : int)  
: int

Return the width of the string will take when displayed with a font.

## Font Enumeration

**Font.GetName** : string

Return the next font name.

## Font.GetSize : int

Return the next size of the font.

## Font.GetStyle (*fontName* : string, var *bold, italic, underline* : boolean)

Get the available styles for a font.

## Font.StartName

Prepare to start listing the names of fonts on the system.

## Font.StartSize (*fontName* : string, *fontStyle* : string)

Prepare to start listing the sizes of a font and style.

## GUI

**See Chapter 5 for a description of the GUI module.**

## Input

**getch (var *ch* : char)**

Get a single character from the keyboard.

**getchar : char**

Return the next keystroke in the keyboard buffer.

**hasch : boolean**

Return true if there is a keystroke in the keyboard buffer.

**Input.Pause**

Wait for any keystroke.



## Joystick

**Joystick.GetInfo** (*joystick* : **int**, **var** *xPos*, *yPos* : **int**, *btn1Pressed*, *btn2Pressed* : **boolean**)

Read the value and button status of a joystick.

## Limits

**maxint** : **int**

Return maximum int type value.

**maxnat** : **nat**

Return maximum nat type value.

**minint** : **int**

Return minimum int type value.

## **minnat : nat**

Return minimum nat type value.

## **Limits.DefaultEW : int**

Return the default exponent width used in printing using "put".

## **Limits.DefaultFW : int**

Return the default fraction width used in printing using "put".

## **Limits.GetExp ( $f$ : real) : int**

Return the (base radix) exponent of  $f$ .

## **Limits.MaxExp : int**

Return the largest (base radix) exponent allowed.

## **Limits.MinExp : int**

Return the smallest (base radix) exponent allowed.

## **Limits.NumDigits : int**

Return the number of radix digits in a real number.

## **Limits.Radix : int**

Return the "radix" (usually 2) of real numbers.

## **Limits.Rreb : real**

Return the relative round-off error bound.

## **Limits.SetExp ( $f$ : real, $e$ : int) : real**

Return the value of  $f$  with the (base radix) exponent replaced.

### **Math**

## **abs ( $x$ : int) : int**

## **abs ( $x$ : real) : real**

Return the absolute value of  $x$ .

## **arctan ( $x : \text{real}$ ) : real**

Return the arctangent with result in radians.

## **arctand ( $x : \text{real}$ ) : real**

Return the arctangent with result in degrees.

## **cos ( $angle : \text{real}$ ) : real**

Return the cosine of angle in radians.

### **Math** (cont...)

## **cosd ( $angle : \text{real}$ ) : real**

Return the cosine of angle in degrees.

## **exp ( $r : \text{real}$ ) : real**

Return the exponentiation function  $e^r$ .

## **ln ( $r : \text{real}$ ) : real**

Return the natural logarithm function  $\ln_e r$ .

**max** ( $x, y : \text{int}$ ) : **int**

**max** ( $x, y : \text{nat}$ ) : **nat**

**max** ( $x, y : \text{real}$ ) : **real**

Return the maximum value of  $x$  and  $y$ .

**min** ( $x, y : \text{int}$ ) : **int**

**min** ( $x, y : \text{real}$ ) : **real**

**min** ( $x, y : \text{nat}$ ) : **nat**

Return the minimum value of  $x$  and  $y$ .

**sign** ( $r : \text{real}$ ) : **-1 .. 1**

Return the sign of the argument.

**sin** (*angle* : real) : real

Return the sine of *angle* in radians.

**sind** (*angle* : real) : real

Return the sine of *angle* in degrees.

**sqrt** (*r* : real) : real

Return the square root of *r*.

## **Mouse**

**Mouse.ButtonChoose** (*choice* : string)

Select the mode for the mouse (either single or multiple button).

**Mouse.ButtonMoved** (*motion* : string) :  
**boolean**

Return whether the mouse button has been pressed.

## **Mouse.ButtonWait** (*motion* : **string**, **var** *x, y, buttonnumber, buttonupdown* : **int**)

Get information about a mouse button being pressed.

## **Mouse.Hide**

Hide the mouse cursor.

## **Mouse.Show**

Show the mouse cursor.

## **Mouse.Where** (**var** *x, y, button* : **int**)

Get the current location of the mouse cursor and the status of the mouse buttons.

## **Music**

## **Music.Play** (*music* : **string**)

Play a series of notes.

## **Music.PlayFile** (*pathName* : **string**)

Play music from a file. File must be in an allowable format.

## **Music.Sound** (*frequency, duration* : **int**)

Play a specified frequency for a specified duration.

## **Music.SoundOff**

Immediately terminate any sound playing.

## **Net**

## **Net.BytesAvailable** (*netStream* : **int**) : **int**

Return the number of bytes available to be read from a net stream.

## **Net.CharAvailable** (*netStream* : **int**) : **boolean**

Return true if there is a character available to be read from a net stream.



## **Net.CloseConnection** (*netStream* : int)

Close a specified connection.

## **Net.HostAddressFromName** (*hostName* : string) : string

Return a host's name given its address.

## **Net.HostNameFromAddress** (*hostAddr* : string) : string

Return a host's address given its host name.

## **Net.LineAvailable** (*netStream* : int) : boolean

Return true if there is a line of text available to be read from a net stream.

## **Net.LocalAddress** : string

Return the host name of the local machine.

## **Net.LocalName : string**

Return the TCP/IP address of the local machine.

## **Net.OpenConnection (*netAddr* : string, *port* : int) : int**

Open a connection to a specified machine.

## **Net.OpenURLConnection (*urlAddr* : string) : int**

Open a connection to a file specified by a URL.

## **Net.TokenAvailable (*netStream* : int) : boolean**

Return true if there is a token available to be read from a net stream.

## **Net.WaitForConnection (*port* : int, var *netAddr* : string) : int**

Wait until a client connects to a specified port.

## PC

### **PC.InPort** (*portAddress* : **nat2**) : **nat1**

Returns a nat1 (byte) value from a specified PC port.

### **PC.InPortWord** (*portAddress* : **nat2**) : **nat2**

Returns a nat2 (word) value from a specified PC port.

### **PC.Interrupt** (*interrupt* : **nat2**, **var** *eax*, *ebx*, *ecx*, *edx*, *esi*, *edi*, *cflag* : **nat**)

Calls a PC interrupt specifying the registers to be passed to the interrupt handler.

### **PC.InterruptSegs** (*interrupt* : **nat2**, **var** *eax*, *ebx*, *ecx*, *edx*, *esi*, *edi*, *cflag*, *ds*, *es*, *ss*, *cs* : **nat**)

Calls a PC interrupt specifying the segments and registers to be passed to the interrupt handler.

## **PC.OutPort** (*portAddress* : **nat2**, *value* : **nat1**)

Sends a nat1 (byte) value to a specified PC port.

## **PC.OutPortWord** (*portAddress* : **nat2**, *value* : **nat2**)

Sends a nat2 (word) value to a specified PC port.

## **PC.ParallelGet** (*port* : **int**) : **nat1**

Returns the value of the pins set on the parallel port.

## **PC.ParallelPut** (*port* : **int**, *value* : **int**)

Sets the values of the pins on the parallel port.

## **PC.SerialGet** (*port* : **int**) : **nat1**

Reads a character from the serial port.

## **PC.SerialHasch** (*port* : **int**) : **boolean**

Returns whether there's a character waiting on the serial port.

## **PC.SerialPut** (*port* : **int**, *value* : **int**)

Sends a character to the serial port.

### **Pic**

## **Pic.Draw** (*picID*, *x*, *y*, *mode* : **int**)

Draw a picture on the screen at location (*x*, *y*).

## **Pic.Free** (*picID* : **int**)

Free up a picture created by **Pic.New** or **Pic.FileNew**.

## **Pic.FileNew** (*pathName* : **string**) : **int**

Read a picture in from a file.

## **Pic.New** (*x1*, *y1*, *x2*, *y2* : **int**) : **int**

Create a picture from a portion of the screen.

## **Pic.Save** (*picID* : **int**, *pathName* : **string**)

Save a picture to a file for use with **Pic.FileNew** or **Pic.ScreenLoad**.

## **Pic.ScreenLoad** (*fileName* : **string**, *x*, *y*, *mode* : **int**)

Load a picture stored in a file straight to the screen.

### **Pic** (cont...)

## **Pic.ScreenSave** (*x1*, *y1*, *x2*, *y2* : **int**, *pathName* : **string**)

Save a portion of the screen to a file for use with **Pic.FileNew** or **Pic.ScreenLoad**.

### **Rand**

## **Rand.Int** (*low*, *high* : **int**) : **int**

Return a random integer from *low* to *high* inclusive.

## **Rand.Next** (*seq* : 1 .. 10) : real

Return a random real number from 0.0 to 1.0 from a sequence.

## **Rand.Real** : real

Return a random real number from 0.0 to 1.0.

## **Rand.Reset**

Set the random seed in the default sequence to the default value.

## **Rand.Seed** (*seed* : nat4, *seq* : 1 .. 10)

Set the random seed in a sequence.

## **Rand.Set** (*i* : nat4)

Set the random seed in the default sequence.

## RGB

**maxcolor : int**

**maxcolour : int**

Return the maximum color number available.

**RGB.AddColor** (*redComp*, *greenComp*,  
*blueComp* : **real**) : **int**

**RGB.AddColour** (*redComp*,  
*greenComp*, *blueComp* : **real**) : **int**

Create a new color with specified red, green, and blue values.



**RGB.GetColor** (*colorNumber* : **int**, **var**  
*redComp*, *greenComp*, *blueComp* : **real**)

**RGB.GetColour** (*colorNumber* : **int**, **var**  
*redComp*, *greenComp*, *blueComp* : **real**)

Get the red, green, and blue values for a color.

**RGB.SetColor** (*colorNumber* : **int**,  
*redComp*, *greenComp*, *blueComp* : **real**)

**RGB.SetColour** (*colorNumber* : **int**,  
*redComp*, *greenComp*, *blueComp* : **real**)

Set the red, green, and blue values for a color.

## Color Names

**black, blue, green, cyan, red, magenta,  
purple, brown, white,**

**gray, grey, brightblue, brightgreen,  
brightcyan, brightred,**

**brightmagenta, brightpurple, yellow**

Names of colors

**brightwhite**

Exists only under DOS. Under Windows, X-Windows and Mac, this is **darkgray**.

**RGB** (cont...)

## **darkgray, darkgrey**

Exists only under Windows, X-Windows and Mac. Under DOS, this is **brightwhite**.

## **colorfg, colourfg**

Foreground color. Under DOS, this is **white**. Under Windows, X-Windows and Mac, this is **black**.

## **colourbg, colourbg**

Background color. Under DOS, this is **black**. Under Windows, X-Windows and Mac, this is **white**.

## **Sprite**

### **Sprite.Animate** (*spriteID*, *picID*, *x*, *y* : **int**, *centered* : **int**)

Change the location and the picture associated with a sprite. Used for animating a moving changing image.

## **Sprite.ChangePic** (*spriteID*, *picID* : int)

Change the picture associated with a sprite.

## **Sprite.Free** (*spriteID* : int)

Dispose of a sprite and free up its memory.

## **Sprite.Hide** (*spriteID* : int)

Hide a visible sprite.

## **Sprite.New** (*picID* : int) : int

Create a new sprite from a picture.

## **Sprite.SetHeight** (*spriteID*, *newHeight* : int)

Set the height of a sprite. Sprites with a greater height appear above sprites with a lesser height. The background is considered height 0. The height may be negative.

## **Sprite.SetPosition** (*spriteID*, *x*, *y* : **int**, *centered* : **boolean**)

Set the location of the sprite. Can specify the center of the sprite or the lower-left corner.

## **Sprite.Show** (*spriteID* : **int**)

Show a previously hidden sprite.

## **Str**

## **index** (*s*, *patt* : **string**) : **int**

Return the position of *patt* within string *s*.

## **length** (*s* : **string**) : **int**

Return the length of the string.

## **repeat** (*s* : **string**, *i* : **int**) : **string**

Return the string *s* concatenated *i* times.

## **Stream**

**eof** (*stream* : **int**) : **boolean**

Return if end of file of a stream has been reached.

**Stream.Flush** (*stream* : **int**)

Flush an output stream.

**Stream.FlushAll**

Flush all open output streams.

## **Sys**

**Sys.GetEnv** (*symbol* : **string**) : **string**

Return the environment string.

**Sys.GetPid** : **int**

Return the process id number of the current task.

## **Sys.Exec** (*command* : string) : int

Execute a program using the operating system.

## Command Line Arguments

## **Sys.FetchArg** (*i* : int) : string

Return the specified command line arg.

## **Sys.Nargs** : int

Return the number of command line args.

**Text**

**maxcol : int**

Return the number of screen text columns.

**maxrow : int**

Return the number of screen text rows.

**Text.Cls**

Clear the screen, setting it to all spaces.

**Text.Color (*clr* : int)**

**Text.Colour (*clr* : int)**

Set text colour.



**Text.ColorBack** (*clr* : int)

**Text.ColourBack** (*clr* : int)

Set text background colour.

**Text.Locate** (*row, col* : int)

Place cursor at character position (*row,col*).

**Text.LocateXY** (*x, y* : int)

Place cursor as close to pixel position (*x,y*) as possible.

**Text** (cont...)

**Text.WhatChar** (*row, col* : int, **var** *ch* :  
**char, var** *foreColor, backColor* : int)

Return the character and text colors at a cursor position.

## **Text.WhatCol : int**

Return the current cursor column.

## **Text.WhatColor : int**

## **Text.WhatColour : int**

Return the currently-active text color.

## **Text.WhatColorBack : int**

## **Text.WhatColourBack : int**

Return the currently-active text background color

## **Text.WhatRow : int**

Return the current cursor row.

## Time

### **Time.Date : string**

Return the current date and time as a string.

### **Time.DateSec (*dateString* : string) : int**

Convert a date/time string into a number of seconds.

### **Time.Delay (*duration* : int)**

Sleep for a specified number of milliseconds.

### **Time.Elapsed : int**

Return milliseconds since the program started to run.

### **Time.ElapsedCPU : int**

Return milliseconds of CPU time since the program started to run.

## **Time.PartsSec** (*year, month, day, hour, minute, second* : **int**) : **int**

Convert the year, month, day, hour, minute, and seconds integers into the number of seconds since 1/1/1970 00:00:00 GMT.

## **Time.Sec** : **int**

Return number of seconds since 1/1/1970 00:00:00 GMT.

## **Time.SecDate** (*timeInSecs* : **int**) : **string**

Convert the number of seconds into a date/time string.

## **Time.SecParts** (*timeInSecs* : **int**, **var** *year, month, day, dayOfWeek, hour, min, sec* : **int**)

Convert the number of seconds since 1/1/1970 00:00:00 GMT into a year, month, day, day of week, hour, minute, and seconds.

**Typeconv**

## From Int

**intreal** ( $i : \text{int}$ ) : **real**

Convert an integer to a real.

**intstr** ( $i : \text{int}$ ) : **string**

Convert an integer to a string.

## From Nat

**natreal** ( $n : \text{nat}$ ) : **real**

Convert a natural number to a real.

**natstr** ( $n : \mathbf{nat}$ ) : **string**

Convert a natural number to a string.

## From Real

**ceil** ( $r : \mathbf{real}$ ) : **int**

Convert a real to an integer (rounding up).

**floor** ( $r : \mathbf{real}$ ) : **int**

Convert a real to an integer (rounding down).

**erealstr** ( $r : \mathbf{real}$ ,  $width$ ,  $fractionWidth$ ,  
 $exponentWidth : \mathbf{int}$ ) : **string**

Convert a real to a string (exponential notation).

**frealstr** (*r* : **real**, *width*, *fractionWidth* : **int**) : **string**

Convert a real to a string (no exponent).

**realstr** (*r* : **real**, *width* : **int**) : **string**

Convert a real to a string.

**round** (*r* : **real**) : **int**

Convert a real to an integer (rounding to closest).

## From String

**strint** (*s* : **string** [, *base* : **int**] ) : **int**

Convert a string to an integer.

**strintok** ( $s : \text{string}$  [,  $base : \text{int}$ ] ) :  
**boolean**

Return whether a string can be converted to an integer.

**strnat** ( $s : \text{string}$  [,  $base : \text{int}$ ] ) : **nat**

Convert a string to a natural number.

**strnatok** ( $s : \text{string}$  [,  $base : \text{int}$ ] ) :  
**boolean**

Return whether a string can be converted to a natural number.

**strreal** ( $s : \text{string}$ ) : **real**

Convert a string to a real.

**strrealok** ( $s : \text{string}$ ) : **boolean**

Return whether a string can legally be converted to a real.



**Typeconv** (cont...)

## To/From ASCII

**chr** (*i* : int) : char

Return a character with the specified ASCII value.

**ord** (*ch* : char) : int

Return the ASCII value of a specified character.

**View**

**maxx** : int

Return the maximum x co-ordinate (width - 1).

**maxy** : int

Return the maximum y co-ordinate (height - 1).

## **View.ClipAdd (*x1*, *y1*, *x2*, *y2* : int)**

Add another rectangle to the clipping region.

## **View.ClipOff**

Stop all clipping

## **View.ClipSet (*x1*, *y1*, *x2*, *y2* : int)**

Set the clipping region to the specified rectangle.

## **View.Set (*setString* : string)**

Change the configuration of the screen.

## **View.WhatDotColor (*x*, *y* : int) : int**

## **View.WhatDotColour (*x*, *y* : int) : int**

Return the color of the pixel at location (*x*, *y*).

## **Window**

### **Window.Close** (*winID* : int)

Close an execution window.

### **Window.GetActive** : int

Return the currently-active execution window.

### **Window.GetPosition** (*winID* : int, var *x*, *y* : int)

Get the current position of an execution window.

### **Window.GetSelect** : int

Return the currently-selected execution window.

### **Window.Hide** (*winID* : int)

Hide an execution window.

## **Window.Open** (*setUpString* : **string**) : **int**

Open a new execution window.

## **Window.Select** (*winID* : **int**)

Select an execution window for output.

**Window** (cont...)

## **Window.Set** (*winID* : **int**, *setUpString* : **string**)

Set the configuration of an execution window.

## **Window.SetActive** (*winID* : **int**)

Select and activate (make front-most) an execution window.

## **Window.SetPosition** (*winID* : **int**, *x*, *y* : **int**)

Set the screen position of an execution window.

## **Window.Show** (*winID* : **int**)

Show an execution window.

### **Miscellaneous Functions not in modules**

#### **Addresses and Sizes**

## **addr** (*reference*)

return the machine address of the item; dirty (implementation-dependent)

## **sizeof** (*item*)

return the number of bytes of the item; dirty (implementation-dependent)

#### **Arrays**

## **lower** (*array* [ , *dimension* ]) : **int**

return the lower bound of array in specified dimension

## **upper** (*array* [ , *dimension* ]) : **int**

return the upper bound of array in specified dimension

#### **Bit Manipulation**

## **bits** (*expn*, *subrange*) : **nat**

extracts a subrange of bits from a natural number expression; the subrange can be the name of a subrange type, an explicit subrange such as 3..7, or a single number such as 9 meaning 9..9.

## Classes

### **objectclass** (*classPointerExpn*)

return the class of the object located by the pointer

### **self**

return a pointer to the current object; this can be used only inside a class

## Enumerated Types

### **pred** (*enumeratedValue*) : *enumeratedValue*

### **pred** (*i* : **int**) : **int**

return the argument minus one or the previous value in the enumerated sequence

### **succ** (*enumeratedValue*) : *enumeratedValue*

### **succ** (*i* : **int**) : **int**

return the argument plus one or the next value in the enumerated sequence

## Miscellaneous Functions (cont...)

## Type Cheats

**cheat** (*targetType*, *expn* [ : *sizeSpec* ])

treats the internal representation of the expression as if it had the target type; the optional size is used to override the size of the expression; dangerous (allows arbitrary access to machine memory)

## Appendix C

### Turing Predefined Subprograms by Category

#### Categories

|                  |     |                    |     |
|------------------|-----|--------------------|-----|
| Type Conversion  | 666 | Random Numbers     | 669 |
| Maximum Numbers  | 667 | Time               | 669 |
| Math             | 667 | Sound              | 669 |
| Strings          | 668 | System             | 670 |
| Enumerated Types | 668 | Character Graphics | 670 |
| Files            | 668 | Pixel Graphics     | 671 |
| Arrays           | 668 | Character Input    | 673 |

#### Type Conversion

#### From Int

**intreal ( $i$  : int) : real**

Convert an integer to a real.

**intstr ( $i$  : int) : string**

Convert an integer to a string.



## From Real

**ceil** ( $r : \text{real}$ ) : **int**

Convert a real to an integer (rounding up).

**erealstr** ( $r : \text{real}$ ,  $width$ ,  $fractionWidth$ ,  
 $exponentWidth : \text{int}$ ) : **string**

Convert a real to a string (exponential notation).

**floor** ( $r : \text{real}$ ) : **int**

Convert a real to an integer (rounding down).

**frealstr** ( $r : \text{real}$ ,  $width$ ,  $fractionWidth : \text{int}$ ) : **string**

Convert a real to a string (no exponent).

**realstr** (*r* : **real**, *width* : **int**) : **string**

Convert a real to a string.

**round** (*r* : **real**) : **int**

Convert a real to an integer (rounding to closest).

## From String

**strint** (*s* : **string** [, *base* : **int**] ) : **int**

Convert a string to an integer.

**strintok** (*s* : **string** [, *base* : **int**] ) :  
**boolean**

Return whether a string can be converted to an integer.

**strreal (*s* : string) : real**

Convert a string to a real.

**strrealok (*s* : string) : boolean**

Return whether a string can legally be converted to a real.

## To/From ASCII

**chr (*i* : int) : char**

Return a character with the specified ASCII value.

**ord (*ch* : char) : int**

Return the ASCII value of a specified character.

## **Maximum Numbers**

**maxint : int**

Return maximum int type value.

**maxnat : nat**

Return maximum nat type value.

## Math

**abs ( $x$  : int) : int**

**abs ( $x$  : real) : real**

Return the absolute value of  $x$ .

**arctan ( $x$  : real) : real**

Return the arctangent with result in radians.

**arctand ( $x$  : real) : real**

Return the arctangent with result in degrees.

**cos ( $angle$  : real) : real**

Return the cosine of angle in radians.

**cosd** (*angle* : real) : real

Return the cosine of angle in degrees.

**exp** (*r* : real) : real

Return the exponentiation function  $e^r$ .

**ln** (*r* : real) : real

Return the natural logarithm function  $\ln_e r$ .

**max** (*x*, *y* : int) : int

**max** (*x*, *y* : real) : real

Return the maximum value of *x* and *y*.

**min** ( $x, y : \text{int}$ ) : **int**

**min** ( $x, y : \text{real}$ ) : **real**

Return the minimum value of  $x$  and  $y$ .

**sign** ( $r : \text{real}$ ) : **-1 .. 1**

Return the sign of the argument.

**sin** ( $angle : \text{real}$ ) : **real**

Return the sine of  $angle$  in radians.

**sind** ( $angle : \text{real}$ ) : **real**

Return the sine of  $angle$  in degrees.

**sqrt** ( $r : \text{real}$ ) : **real**

Return the square root of  $r$ .

## Strings

**index** ( $s, patt : \text{string}$ ) : int

Return the position of *patt* within string *s*.

**length** ( $s : \text{string}$ ) : int

Return the length of the string.

**repeat** ( $s : \text{string}, i : \text{int}$ ) : string

Return the string *s* concatenated *i* times.

## Enumerated Types

**pred** ( $enumeratedValue$ ) : *enumeratedValue*

**pred** ( $i : \text{int}$ ) : int

return the argument minus one or the previous value in the enumerated sequence

**succ** ( $enumeratedValue$ ) : *enumeratedValue*

**succ** ( $i : \text{int}$ ) : int

return the argument plus one or the next value in the enumerated sequence

## Files

**eof** (*stream* : **int**) : **boolean**

Return if end of file of a stream has been reached.

## Arrays

**lower** (*array* [ , *dimension* ]) : **int**

return the lower bound of array in specified dimension

**upper** (*array* [ , *dimension* ]) : **int**

return the upper bound of array in specified dimension

## Random Numbers

**rand** (**var** *r* : **real**)

generate a random number from 0 to 1

**randint** (**var** *i* : **int**, *low*, *high* : **int**)

generate a random integer from low to high inclusive



**randnext** (**var** *v* : **real**, *seq* : 1..10)

produce next pseudo-random number in specified sequence

**randomize**

resets random number sequence

**randseed** (*seed* : **int** , *seq* : 1..10)

restarts the specified sequence of pseudo-random numbers with seed

## **Time**

**clock** (**var** *c* : **int**)

time in milliseconds since process began

**date** (**var** *date* : **string**)

date in "DD Mmm YY" format

**sysclock** (**var** *c* : **int**)

CPU time in milliseconds used by process

**time (var *t* : string)**

time in "HH:MM:SS" format

**wallclock (var *c* : int)**

time in seconds since 00:00:00 GMT Jan 1, 1970

## **Sound**

**play (*music* : string)**

play musical notes

**playdone : boolean**

determine if play command finished

**sound (*freq* , *duration* : int)**

play specified sound

## System

**delay** (*duration* : int)

delay program for duration in milliseconds

**fetcharg** (*i* : int) : string

return the specified command line arg

**getenv** (*symbol* : string) : string

return the environment string

**getpid** : int

return the process id

**nargs** : int

return the number of command line args

## **parallelget : int**

read pins of the parallel port

## **parallelput (*val* : int)**

set pins on the parallel port

## **serialget : int**

read a character from the serial port

## **serialput (*val* : int)**

sends a character to the serial port

## **system (*command* : string , var *ret* : int)**

execute operating system command

### **Character Graphics**

## **cls**

clear the screen

**color** (*clr* : int)

**colour** (*clr* : int)

set text colour

**colorback** (*clr* : int)

**colourback** (*clr* : int)

set text background colour

**locate** (*row* , *col* : int)

place cursor at character position (row,col)

**maxcol** : int

return the number of screen text columns

**maxcolor : int**

**maxcolour : int**

return the maximum screen text colour

**maxrow : int**

return the number of screen text rows

**setscreen (*s* : string)**

set the mode of the screen

**whatcol : int**

return the current cursor column

**whatcolor : int**

**whatcolour : int**

return the currently-active text colour

**whatcolorback : int**

**whatcolourback : int**

return the current text background colour

**whatrow : int**

return the current cursor row

**whattextchar : string (1)**

return the character at cursor position

**whattextcolor : int**

**whattextcolour : int**

return the character colour at cursor

**whattextcolorback : int**

**whattextcolourback : int**

return the character background colour at cursor

## **Pixel Graphics**

**cls**

clear the screen



**color** (*clr* : **int**)

**colour** (*clr* : **int**)

set text colour

**colorback** (*clr* : **int**)

**colourback** (*clr* : **int**)

instantly change background colour

**drawarc** (*x*, *y*, *xRadius*, *yRadius*,  
*initialAngle*, *finalAngle*, *clr* : **int**)

draw an arc on screen centred at (x,y)

**drawbox** (*x1* , *y1* , *x2* , *y2* , *clr* : **int**)

draw a box on screen

**drawdot** (*x* , *y* , *clr* : **int**)

draw a dot on screen at (x,y)

**drawfill** (*x* , *y* , *fillColor* , *borderColor* :  
**int**)

fills in a figure of color borderColor

**drawfillarc** (*x* , *y* , *xRadius* , *yRadius* ,  
*initialAngle* , *finalAngle* , *clr* : **int**)

draw a filled pie shape wedge on screen centred at (x,y)

**drawfillbox** (*x1* , *y1* , *x2* , *y2* , *clr* : **int**)

draw a filled box on screen

**drawfilloval** (*x* , *y* , *xRadius* , *yRadius* ,  
*clr* : **int**)

draw a filled oval on screen centred at (x,y)

**drawfillpolygon** (*x* , *y* :array 1..\* of int ,  
*n* : int , *clr* : int)

draw a filled polygon onto the screen

**drawline** (*x1* , *y1* , *x2* , *y2* , *clr* : int)

draw a line on screen

**drawoval** (*x* , *y* , *xRadius* , *yRadius* , *clr* :  
int)

draw an oval on screen centred at (x,y)

**drawpolygon** (*x* , *y* :array 1..\* of int , *n* :  
int , *clr* : int)

draw a polygon onto the screen

**drawpic** (*x* , *y* : int , *buffer* : array 1..\* of  
int , *picmode* : int)

copies a rectangular area from buffer onto the screen

**locate** (*row* , *col* : int)

place cursor at character position (row, col)

**locatexy** (*x* , *y* : int)

place cursor at pixel position (x,y)

**maxcol** : int

return the number of screen text columns

**maxcolor** : int

**maxcolour** : int

return the maximum screen colour

**maxrow** : int

return the number of screen text rows

**maxx : int**

return the maximum x pixel value

**maxy : int**

return the maximum y pixel value

**palette (*p* : int)**

sets the colour palette to use

**setscreen (*s* : string)**

set the mode of the screen

**sizepic (*x1* , *y1* , *x2* , *y2* : int) : int**

determine the necessary size of a buffer to hold area from screen

**takepic (*x1* , *y1* , *x2* , *y2* : int , var *buffer*  
: array 1..\* of int)**

copies a rectangular area from screen into buffer

**whatcol : int**

return the current cursor column

**whatcolor : int**

**whatcolour : int**

return the currently active text colour

**whatcolorback : int**

**whatcolourback : int**

return the current background colour

**whatdotcolor ( $x, y : \text{int}$ ) : int**

**whatdotcolour ( $x, y : \text{int}$ ) : int**

return the colour of pixel at (x,y)

## **whatpalette : int**

return the current palette number

## **whatrow : int**

return the current cursor row

### **Character Input**

## **getch (var *ch* : char)**

get a single character from the keyboard

## **hasch : boolean**

return true if there is a keystroke in the keyboard buffer

# Appendix D

## Operators

### Mathematical Operators

| <u>Operator</u> | <u>Operation</u>      | <u>Result Type</u> |
|-----------------|-----------------------|--------------------|
| Prefix <b>+</b> | Identity              | As Operands        |
| Prefix <b>-</b> | Negative              | As Operands        |
| <b>+</b>        | Addition              | As Operands        |
| <b>-</b>        | Subtraction           | As Operands        |
| <b>*</b>        | Multiplication        | As Operands        |
| <b>/</b>        | Division              | As Operands        |
| <b>div</b>      | Integer Division      | <b>int</b>         |
| <b>mod</b>      | Modulo                | <b>int</b>         |
| <b>rem</b>      | Remainder             | <b>int</b>         |
| <b>**</b>       | Exponentiation        | As Operands        |
| <b>&lt;</b>     | Less Than             | <b>boolean</b>     |
| <b>&gt;</b>     | Greater Than          | <b>boolean</b>     |
| <b>=</b>        | Equals                | <b>boolean</b>     |
| <b>&lt;=</b>    | Less Than or Equal    | <b>boolean</b>     |
| <b>&gt;=</b>    | Greater Than or Equal | <b>boolean</b>     |
| <b>not=</b>     | Not Equal             | <b>boolean</b>     |

### Boolean Operators

| <u>Operator</u>   | <u>Operation</u> | <u>Result Type</u> |
|-------------------|------------------|--------------------|
| Prefix <b>not</b> | Negation         | <b>boolean</b>     |
| <b>and</b>        | And              | <b>boolean</b>     |
| <b>or</b>         | Or               | <b>boolean</b>     |
| <b>xor</b>        | Exclusive Or     | <b>boolean</b>     |
| <b>=&gt;</b>      | Implication      | <b>boolean</b>     |

### Set Operators

| <u>Operator</u> | <u>Operation</u> | <u>Result Type</u> |
|-----------------|------------------|--------------------|
| <b>+</b>        | Union            | <b>set</b>         |
| <b>-</b>        | Set Subtraction  | <b>set</b>         |
| <b>*</b>        | Intersection     | <b>set</b>         |
| <b>=</b>        | Equality         | <b>boolean</b>     |
| <b>not=</b>     | Inequality       | <b>boolean</b>     |



|              |                          |                |
|--------------|--------------------------|----------------|
| <b>&lt;=</b> | Subset                   | <b>boolean</b> |
| <b>&lt;</b>  | Strict (Proper) Subset   | <b>boolean</b> |
| <b>&gt;=</b> | Superset                 | <b>boolean</b> |
| <b>&gt;</b>  | Strict (Proper) Superset | <b>boolean</b> |

#### Operators on Members and Sets

| <u>Operator</u> | <u>Operation</u>  | <u>Result Type</u> |
|-----------------|-------------------|--------------------|
| <b>in</b>       | Member of Set     | <b>boolean</b>     |
| <b>not in</b>   | Not Member of Set | <b>boolean</b>     |
| <b>xor</b>      | Exclusive Or      | <b>set</b>         |

#### Bit Manipulation Operators

| <u>Operator</u> | <u>Operation</u>      | <u>Result Type</u> |
|-----------------|-----------------------|--------------------|
| <b>shl</b>      | Shift left            | <b>nat</b>         |
| <b>shr</b>      | Shift right           | <b>nat</b>         |
| <b>and</b>      | Bit-wise And          | <b>nat</b>         |
| <b>or</b>       | Bit-wise Or           | <b>nat</b>         |
| <b>xor</b>      | Bit-wise Exclusive Or | <b>nat</b>         |

#### Pointer Operators

| <u>Operator</u> | <u>Operation</u> | <u>Result Type</u> |
|-----------------|------------------|--------------------|
| <b>^</b>        | Follow pointer   | Target type        |

#### Type Cheats

| <u>Operator</u> | <u>Operation</u> | <u>Result Type</u> |
|-----------------|------------------|--------------------|
| <b>#</b>        | Type cheat       | <b>nat</b>         |

#### Operator Short Forms

These can be used in place of the above notation.

|               |              |
|---------------|--------------|
| <b>not</b>    | <b>~</b>     |
| <b>not=</b>   | <b>~=</b>    |
| <b>not in</b> | <b>~in</b>   |
| <b>and</b>    | <b>&amp;</b> |
| <b>or</b>     | <b> </b>     |

#### Operator Precedence

Highest precedence operators first.

(1) **\*\* , ^ , #**

- (2) prefix + and -
- (3) \*, /, **div**, **mod**, **rem**, **shl**, **shr**
- (4) +, -, **xor**
- (5) <, >, =, <=, >=, **not=**, **in**, **not in**
- (6) **not**
- (7) **and**
- (8) **or**
- (9) =>

## Appendix E

### File Statements

#### File Commands

|              |                                        |
|--------------|----------------------------------------|
| <b>open</b>  | open a file                            |
| <b>close</b> | close a file                           |
| <b>put</b>   | write alphanumeric text to a file      |
| <b>get</b>   | read alphanumeric text from a file     |
| <b>write</b> | binary write to a file                 |
| <b>read</b>  | binary read from a file                |
| <b>seek</b>  | move to a specified position in a file |
| <b>tell</b>  | report the current file position       |
| <b>eof</b>   | check for end of file                  |

#### File Command Syntax

**open** : *streamNo*, *fileName*, *ioCapability* {, *ioCapability* }

**ioCapability** is one of **get**, **put**, **read**, **write**, **seek**, **mod**

**put** or **write** capability will cause any existing file to be truncated to zero length unless the **mod** capability is also specified.

**seek** capability is needed to use **seek** or **tell** commands.

**close** : *streamNo*

**get** : *streamNo* , *getItem* { , *getItem* }

**put** : *streamNo* , *putItem* { , *putItem* }

**read** : *streamNo* [ : *fileStatus* ] , *readItem* { , *readItem* }

**write** : *streamNo* [ : *fileStatus* ] , *writeItem* { , *writeItem* }

**seek** : *streamNo* , *filePosition* or **seek** : *streamNo* , \*

**tell** : *streamNo* , *filePositionVar*

**eof** ( *streamNo* ) : **boolean** (This is a function)

## Appendix F

### Control Constructs

**FOR**      **for** [ **decreasing** ] *variable* : *startValue* .. *endValue*  
                  ... statements ...  
                  **exit when** *expn*  
                  ... statements ...  
          **end for**

**LOOP**     **loop**  
                  ... statements ...  
                  **exit when** *expn*  
                  ... statements ...  
          **end loop**

**IF**        **if** condition **then**  
                  ... statements ...  
          { **elsif** condition **then**  
                  ... statements ... }  
          [ **else**  
                  ... statements ... ]  
          **end if**

**CASE**     **case** *expn* **of**  
                  { **label** *expn* { , *expn* } :  
                          ... statements ... }  
                  [ **label** :  
                          ... statements ... ]  
          **end case**

Any number of **exit** and **exit when** constructs can appear at any place inside **for .. end for** constructs and **loop .. end loop** constructs.

## Appendix G

### IBM PC Keyboard Codes

The ASCII value of the character  
returned by getch

|               |    |         |    |   |    |         |     |
|---------------|----|---------|----|---|----|---------|-----|
|               | 0  | (space) | 32 | @ | 64 | `       | 96  |
| Ctrl-A        | 1  | !       | 33 | A | 65 | a       | 97  |
| Ctrl-B        | 2  | "       | 34 | B | 66 | b       | 98  |
|               | 3  | #       | 35 | C | 67 | c       | 99  |
| Ctrl-D        | 4  | \$      | 36 | D | 68 | d       | 100 |
| Ctrl-E        | 5  | %       | 37 | E | 69 | e       | 101 |
| Ctrl-F        | 6  | &       | 38 | F | 70 | f       | 102 |
| Ctrl-G        | 7  | '       | 39 | G | 71 | g       | 103 |
| Ctrl-H / BS   | 8  | (       | 40 | H | 72 | h       | 104 |
| Ctrl-I / TAB  | 9  | )       | 41 | I | 73 | i       | 105 |
| Ctrl-J/ Enter | 10 | *       | 42 | J | 74 | j       | 106 |
| Ctrl-K        | 11 | +       | 43 | K | 75 | k       | 107 |
| Ctrl-L        | 12 | ,       | 44 | L | 76 | l       | 108 |
| Ctrl-M        | 13 | -       | 45 | M | 77 | m       | 109 |
| Ctrl-N        | 14 | .       | 46 | N | 78 | n       | 110 |
| Ctrl-O        | 15 | /       | 47 | O | 79 | o       | 111 |
| Ctrl-P        | 16 | 0       | 48 | P | 80 | p       | 112 |
| Ctrl-Q        | 17 | 1       | 49 | Q | 81 | q       | 113 |
| Ctrl-R        | 18 | 2       | 50 | R | 82 | r       | 114 |
| Ctrl-S        | 19 | 3       | 51 | S | 83 | s       | 115 |
| Ctrl-T        | 20 | 4       | 52 | T | 84 | t       | 116 |
| Ctrl-U        | 21 | 5       | 53 | U | 85 | u       | 117 |
| Ctrl-V        | 22 | 6       | 54 | V | 86 | v       | 118 |
| Ctrl-W        | 23 | 7       | 55 | W | 87 | w       | 119 |
| Ctrl-X        | 24 | 8       | 56 | X | 88 | x       | 120 |
| Ctrl-Y        | 25 | 9       | 57 | Y | 89 | y       | 121 |
| Ctrl-Z        | 26 | :       | 58 | Z | 90 | z       | 122 |
| Ctrl-[ / ESC  | 27 | ;       | 59 | [ | 91 | {       | 123 |
| Ctrl-\        | 28 | <       | 60 | \ | 92 |         | 124 |
| Ctrl-]        | 29 | =       | 61 | ] | 93 | }       | 125 |
| Ctrl-^        | 30 | >       | 62 | ^ | 94 | ~       | 126 |
| Ctrl-_        | 31 | ?       | 63 | _ | 95 | Ctrl-BS | 127 |

|                  |     |       |     |             |     |             |     |
|------------------|-----|-------|-----|-------------|-----|-------------|-----|
| Alt-9            | 128 | Alt-D | 160 | F6          | 192 | Ctrl-F3     | 224 |
| Alt-0            | 129 | Alt-F | 161 | F7          | 193 | Ctrl-F4     | 225 |
| Alt--            | 130 | Alt-G | 162 | F8          | 194 | Ctrl-F5     | 226 |
| Alt-=            | 131 | Alt-H | 163 | F9          | 195 | Ctrl-F6     | 227 |
| Ctrl-PgUp        | 132 | Alt-J | 164 | F10         | 196 | Ctrl-F7     | 228 |
| *F11             | 133 | Alt-K | 165 |             | 197 | Ctrl-F8     | 229 |
| *F12             | 134 | Alt-L | 166 |             | 198 | Ctrl-F9     | 230 |
| *Shift-F11       | 135 |       | 167 | Home        | 199 | Ctrl-F10    | 231 |
| *Shift-F12       | 136 |       | 168 | Up Arrow    | 200 | Alt-F1      | 232 |
| *Ctrl-F11        | 137 |       | 169 | PgUp        | 201 | Alt-F2      | 233 |
| *Ctrl-F12        | 138 |       | 170 |             | 202 | Alt-F3      | 234 |
| *Alt-F11         | 139 |       | 171 | Left Arrow  | 203 | Alt-F4      | 235 |
| *Alt-F12         | 140 | Alt-Z | 172 |             | 204 | Alt-F5      | 236 |
| *Ctrl-Up Arrow   | 141 | Alt-X | 173 | Right Arrow | 205 | Alt-F6      | 237 |
|                  | 142 | Alt-C | 174 |             | 206 | Alt-F7      | 238 |
| Back TAB         | 143 | Alt-V | 175 | End         | 207 | Alt-F8      | 239 |
| Alt-Q            | 144 | Alt-B | 176 | Dn Arrow    | 208 | Alt-F9      | 240 |
| *Ctrl-Down Arrow | 145 | Alt-N | 177 | PgDn        | 209 | Alt-F10     | 241 |
| Alt-W            |     |       |     |             |     |             |     |
| *Ctrl-Insert     | 146 | Alt-M | 178 | Insert      | 210 |             | 242 |
| Alt-E            |     |       | 179 | Delete      | 211 | Ctrl-L Arrw | 243 |
| *Ctrl-Del ete    | 147 |       | 180 |             |     | Ctrl-R Arrw | 244 |
| Alt-R            |     |       | 181 | Shift-F1    | 212 | Ctrl-End    | 245 |
| Alt-T            | 148 |       | 182 | Shift-F2    | 213 | Ctrl-PgDn   | 246 |
| Alt-Y            | 149 |       | 183 | Shift-F3    | 214 | Ctrl-Home   | 247 |
| Alt-U            | 150 |       | 184 | Shift-F4    | 215 | Alt-1       | 248 |
| Alt-I            | 151 |       | 185 | Shift-F5    | 216 | Alt-2       | 249 |
| Alt-O            | 152 |       | 186 | Shift-F6    | 217 | Alt-3       | 250 |
| Alt-P            | 153 |       | 187 | Shift-F7    | 218 | Alt-4       | 251 |
|                  | 154 | F1    | 188 | Shift-F8    | 219 | Alt-5       | 252 |
|                  | 155 | F2    | 189 | Shift-F9    | 220 | Alt-6       | 253 |
|                  | 156 | F3    | 190 | Shift-F10   | 221 | Alt-7       | 254 |
|                  | 157 | F4    | 191 | Ctrl-F1     | 222 | Alt-8       | 255 |
| Alt-A            | 158 | F5    |     | Ctrl-F2     | 223 |             |     |
| Alt-S            | 159 |       |     |             |     |             |     |

\* = OOT Only keystroke

Ctrl-@, Ctrl-C and Ctrl-Break will terminate a Turing Program

## Appendix H

### IBM PC / Macintosh / ICON Turing Character Set

The ASCII value of each character displayed by Turing

|    |           |    |       |    |   |     |           |
|----|-----------|----|-------|----|---|-----|-----------|
| 0  | Space     | 32 | Space | 64 | @ | 96  | `         |
| 1  | □         | 33 | !     | 65 | A | 97  | a         |
| 2  | □         | 34 | "     | 66 | B | 98  | b         |
| 3  | □         | 35 | #     | 67 | C | 99  | c         |
| 4  | □         | 36 | \$    | 68 | D | 100 | d         |
| 5  | □         | 37 | %     | 69 | E | 101 | e         |
| 6  | not shown | 38 | &     | 70 | F | 102 | f         |
| 7  |           | 39 | '     | 71 | G | 103 | g         |
| 8  | Backspace | 40 | (     | 72 | H | 104 | h         |
| 9  | Tab       | 41 | )     | 73 | I | 105 | i         |
| 10 | Newline   | 42 | *     | 74 | J | 106 | j         |
| 11 | not shown | 43 | +     | 75 | K | 107 | k         |
| 12 | not shown | 44 | ,     | 76 | L | 108 | l         |
| 13 | not shown | 45 | -     | 77 | M | 109 | m         |
| 14 |           | 46 | .     | 78 | N | 110 | n         |
| 15 | □         | 47 | /     | 79 | O | 111 | o         |
| 16 | □         | 48 | 0     | 80 | P | 112 | p         |
| 17 | □         | 49 | 1     | 81 | Q | 113 | q         |
| 18 | □         | 50 | 2     | 82 | R | 114 | r         |
| 19 | □         | 51 | 3     | 83 | S | 115 | s         |
| 20 | □         | 52 | 4     | 84 | T | 116 | t         |
| 21 | □         | 53 | 5     | 85 | U | 117 | u         |
| 22 | □         | 54 | 6     | 86 | V | 118 | v         |
| 23 | □         | 55 | 7     | 87 | W | 119 | w         |
| 24 | □         | 56 | 8     | 88 | X | 120 | x         |
| 25 | □         | 57 | 9     | 89 | Y | 121 | y         |
| 26 | □         | 58 | :     | 90 | Z | 122 | z         |
| 27 | □         | 59 | ;     | 91 | [ | 123 | {         |
| 28 | □         | 60 | <     | 92 | \ | 124 |           |
| 29 | □         | 61 | =     | 93 | ] | 125 | }         |
| 30 | not shown | 62 | >     | 94 | ^ | 126 | ~         |
| 31 | not shown | 63 | ?     | 95 | _ | 127 | not shown |



|     |   |     |   |     |     |     |   |
|-----|---|-----|---|-----|-----|-----|---|
| 128 | Ä | 160 | † | 192 | ı   | 224 | ‡ |
| 129 | Å | 161 | ° | 193 | ı   | 225 | · |
| 130 | Ç | 162 | ¢ | 194 | ¬   | 226 | , |
| 131 | É | 163 | £ | 195 | √   | 227 | „ |
| 132 | Ñ | 164 | § | 196 | f   | 228 | ‰ |
| 133 | Ö | 165 | • | 197 | ≈   | 229 | À |
| 134 | Ü | 166 | ¶ | 198 | Δ   | 230 | Ê |
| 135 | á | 167 | ß | 199 | «   | 231 | Á |
| 136 | à | 168 | ® | 200 | »   | 232 | Ë |
| 137 | â | 169 | © | 201 | ... | 233 | È |
| 138 | ä | 170 | ™ | 202 |     | 234 | Í |
| 139 | ã | 171 | ’ | 203 | À   | 235 | Î |
| 140 | å | 172 | ” | 204 | Å   | 236 | Ï |
| 141 | ç | 173 | ≠ | 205 | Ö   | 237 | İ |
| 142 | é | 174 | Æ | 206 | Œ   | 238 | Ó |
| 143 | è | 175 | Ø | 207 | œ   | 239 | Ô |
| 144 | ê | 176 | ∞ | 208 | —   | 240 | □ |
| 145 | ë | 177 | ± | 209 | —   | 241 | Ò |
| 146 | í | 178 | ≤ | 210 | “   | 242 | Û |
| 147 | ì | 179 | ≥ | 211 | ”   | 243 | Ü |
| 148 | î | 180 | ¥ | 212 | ‘   | 244 | Ù |
| 149 | ï | 181 | μ | 213 | ’   | 245 | ı |
| 150 | ñ | 182 | ∂ | 214 | ÷   | 246 | ^ |
| 151 | ó | 183 | Σ | 215 | ◊   | 247 | ~ |
| 152 | ò | 184 | Π | 216 | ÿ   | 248 | — |
| 153 | ô | 185 | π | 217 | Ÿ   | 249 | ˘ |
| 154 | ö | 186 | ∫ | 218 | /   | 250 | ˙ |
| 155 | õ | 187 | ª | 219 | €   | 251 | ° |
| 156 | ú | 188 | º | 220 | ‹   | 252 | , |
| 157 | ù | 189 | Ω | 221 | ›   | 253 | ” |
| 158 | û | 190 | æ | 222 | fi  | 254 | ˚ |
| 159 | ü | 191 | ø | 223 | fl  | 255 |   |

